1972-12

GRAFTRAN: graphic extensions to FORTRAN

Elkins, David R; Wolf, Edward J.

Monterey, California. Naval Postgraduate School

GRAFTRAN: GRAPHIC EXTENSIONS TO FORTRAN

David R. Elkins

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

GRAFTRAN: GRAPHIC EXTENSIONS TO FORTRAN

by

David R. Elkins

and

Edward J. Wolf

Thesis Advisor: G. L. Barksdale

December 1972

Approved for public release; distribution unlimited.

GRAFTRAN: Graphic Extensions to FORTRAN

by

David R. Elkins
Lieutenant, United States Navy Reserve
B.S., California State Polytechnic College, 1964

and

Edward J. Wolf
Lieutenant, United States Navy
B.S., University of North Dakota, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1972

ABSTRACT

This paper describes a computer graphics language which can be used to easily display complex structures on a medium scale computer system, such as the XDS9300/AGT-10 complex at the Naval Postgraduate School. GRAFTRAN (Graphic Extensions to FORTRAN) provides the full capabilities of FORTRAN and includes features that represent a cross-section of operations which are used in present day computer graphics systems. The user is completely isolated from the mechanism to construct and display an image and the manipulation of the data involved. Also included is a survey of languages and data structures currently used in graphical applications. An example GRAFTRAN program is included.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. PROBLEM DESCRIPTION

In a graphical interactive system, the user sets goals, formulates hypotheses, determines criteria, and performs evaluations while the computer based graphics system does the routine work necessary to prepare the way for insights and decisions by the user. This partnership between man and machine is what J. C. R. Licklider [Ref. 1] calls "Man-Computer Symbiosis."

Without proper software support routines, however, a user of a computer graphic system must necessarily have to familiarize himself with the literature in order to extract and apply to his problem the ideas and principles that were developed in previous applications. Much of his effort is expended in the structuring of his data and program. Frequently, the problems encountered in such structuring are greater than the application problem he set out to solve. This has been the experience of the authors when implementing computer aided design programs on a system which only has a few basic communication subroutines for information transfer between the main computer and the graphics subsystem.

B. USER ORIENTED GRAPHICS LANGUAGE

Ideally, a problem oriented language should be provided to the user so that he can devote the majority of his attention to the application. The language should be descriptive enough that the user does not have to learn an estensive set of rules or be required to have more than a cursory knowledge of the data structuring or the communication involved.

The syntax of the language should reflect the power of graphical information.

C. SCOPE OF THESIS

The purpose of this paper is to survey some of the graphic language implementations, investigate and determine which attributes of graphics are most widely used in graphic languages, and finally to incorporate some of these attributes into a FORTRAN based language.

Section II presents the general characteristics of graphic languages and gives for illustration a description of several graphic language systems. In Section III, the attributes of graphic languages are described and the incorporation of some of the attributes in a FORTRAN based language is presented. The language is called GRAFTRAN (Graphic Extensions to FORTRAN) and a particular implementation for the XDS 9300 computer and an ADAGE AGT-10 graphic subsystem is described in Section IV. Finally, conclusions and recommended future developments are given in Section V. The main body of the document is left as general as possible. Detailed information is relegated to appendices.

II. GRAPHIC LANGUAGES

A. GENERAL CHARACTERISTICS

Interactive systems are intended to make efficient use of both man and machine by allowing a designer (user) to interact with the computer system while his program is running; in this way, he can influence the course of the computation or process involved. With an interactive graphics system, a user of the system should be able to reference any part of one of the displays (and parts within the images) which he has created. This phase of interaction requires that data be structured to allow necessary building, searching and accessing of related data to be performed efficiently. The problems encountered in implementing a powerful data structure are frequently greater than those involved in the application itself. Many languages have been created to provide programmers with the capability to build their own data structures; the programmer (user), of course, is still left with the problem of designing a complex data structure. This is generally undesirable since the application programmer is primarily concerned with application and should not have to be cognizant of the implementation of the data structure. Consequently, several higher level graphics languages have been developed which allow the programmer to concentrate on his problem rather than structural implementation problems.

B. SURVEY OF GRAPHIC LANGUAGE CAPABILITIES

As with many areas in computer application, the literature on computer graphics is extensive. One bibliography alone [Ref. 2] lists over 300 reports.

In this section a brief survey of some developmental efforts in the area of graphic language definition is presented. The discussion is not meant to emphasize these particular languages but rather to give an idea of the diversity of approaches.

1. Developments on the Whirlwind Computer

One of the earliest uses of a Cathode-Ray Tube (CRT) display in conjunction with computer operations was on the Whirlwind Computer at the Massachusetts Institute of Technology in 1956 to facilitate program debugging [Ref. 3].

The Whirlwind CRT display was given moderate input capabilities in the SAGE (Semi-Automatic Ground Environment) air defense system. Light guns (predecessors to today's light pens) were used to allow a operator to direct the computer program to select specific displayed objects as targets.

2. Sketchpad and Sketchpad III

The first significant use of the display console as an interactive computer input/output device was demonstrated by I. E. Sutherland in his Sketchpad system [Ref. 4] in 1963. Sketchpad, through the use of dials, buttons, and light pen, provided an operator with the capability to alter the graphic display or otherwise alter the action of the running program. Besides functions like MOVE, ROTATE, DRAW, ERASE, DELETE, etc., a set of constraint functions were implemented; these include such concepts as making a line perpendicular or parallel to another line or terminating a line when it intersects another line, etc. A hierarchial structure was implemented in which a subpicture can be defined and attachment points designated within it so that subpictures can be joined together to form a new picture.

The following example (extracted from Ref. 4) demonstrates the power of the system, in general, and additionally the data structure and the hierarchy.

Suppose that a user desires to construct a hexagon. By pointing the light pen at the display and depressing a button called "DRAW", the system will construct a straight line segment from the initial position of the pen when the button was depressed to the final position of the pen when the button was released. Repeating this sequence with each endpoint of a line the starting of the next line, an irregular hexagon can be created as shown in Figure 1A.

The hexagon can be made regular by inscribing it in a circle. To draw the circle, the light pen is placed where the center of the circle would be and the button "CIRCLE CENTER" depressed. Then, a point is chosen on the circle (establishing the radius) and the button "DRAW" is depressed. The angular length of the arc is determined by the intersection of the imaginary pen path and an imaginary radial from the center (Figure 1B). Next, a corner of the irregular hexagon is stretched like a rubber band and positioned on the circle using the "MOVE" button (Figure 1C). Continuing in this manner, each corner of the circle is placed at roughly equal spacing. By making the constraints (using constraint functions) that the hexagon be inscribed in the circle and that each side be equal length, a regular hexagon is formed, and the circle erased.

By using this hexagon as a basic element, a more complex structure can be built such as that in Figure 2.

To do so, one would proceed in the following manner. Since a large number of hexagons are to be attached together, each of the six

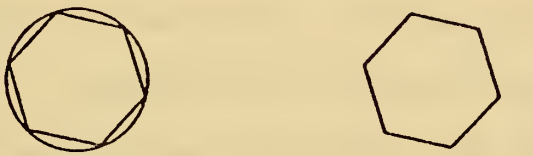


FIGURE 1--CONSTRUCTION OF HEXAGONS

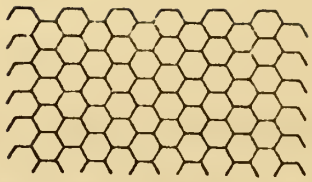


FIGURE 2

HEXAGON PATTERN

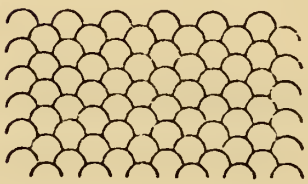


FIGURE 3

SCALLOP PATTERN

(Figures 1-3 were extracted from Ref. 4)

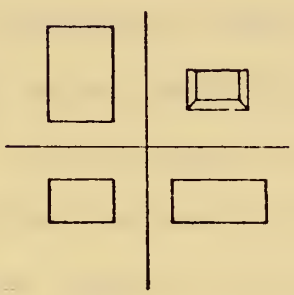


FIGURE 4
FOUR QUADRANT VIEW
BEFORE ROTATION

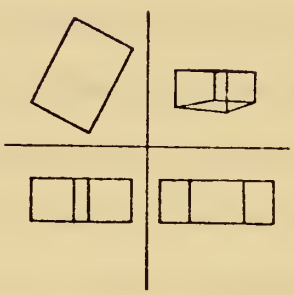


FIGURE 5
FOUR QUADRANT VIEW
AFTER ROTATION

(Figures 4 and 5 were extracted from Ref. 5)

corners is designated as an attachment point. The image is stored away as a subpicture and a new drawing is started. By calling up seven copies of the subpicture "HEXAGON" and using ROTATE, SCALE, etc. buttons, a pattern such as that in Figure 1G can be created. Now, by pointing to the corner of one hexagon, pressing a button, and then pointing to the corner of another hexagon, the two corners can be joined. Similarly, all hexagons can be joined as in Figure 1H. This image can now be designated as a subpicture and copies of it can be joined in a like fashion to that above to form the pattern in Figure 2.

Essentially, in the above example, a structure of subpictures has been created (hierarchy). The structure created is retained even if the basic components are changed (in this case a hexagon). For instance, if the hexagon is changed into a semicircle, the pattern in Figure 3 is the result.

Sketchpad was more a system than a language and specifically tailored to the TX-2 computer.

Sketchpad III [Ref. 5] used the data structure and utility programs that operate on the data structure of the sketchpad system to develop a three-dimensional graphic system. The designer of Sketchpad III (a student in Mechanical Engineering) stated that "the heavy dependence of mechanical design upon three-dimensional objects makes such a facility an indispensible part of the full system which is envisioned."

In the system, four views of an object are displayed; one in each of four quadrants on the cathode ray tube. A perspective view of the object appears in the first quadrant and three orthogonal views appear in the remaining quadrants (second quadrant--top view; third quadrant--front view; fourth quadrant--side view). The objects are

displayed as wire frames. That is, lines that are normally hidden are not obscured as they should be and hence, fail to convey depth information. The perspective gives the illusion of three-dimensions with respect to the orthogonal views. Figure 4 is an example of a display.

Pushbuttons (function switches), at the graphic terminal, provide for such things as erase what the light pen is pointing at, move what the light pen is pointing at according to the movement of the pen, translate the drawing the light pen is pointing at, etc. Potentiometers (variable control dials) are used for such things as 1) magnify or reduce the image, 2) rotate the image clockwise or counterclockwise, or 3) force or relax the perspective (by changing the convergence of the lines). An example of an operation might be to designate the image in quadrant two (Figure 4) with the light pen and adjust the VCD for ROTATE. The final result might be that of Figure 5.

The system also has the capability to draw three dimensional objects with the light pen. In drawing a three dimensional object on a two dimensional surface, depth information must somehow be conveyed. Since a drawing can be rotated to any position in space, lines can be drawn in a three dimensional object by drawing in a plane. An image is rotated until the area in which a line is to be drawn is parallel to a viewing quadrant. The line is then drawn to its true length and position in that plane; the depth coordinate is thus constant as the pen traces the line.

Some of the things that Sketchpad III did not have but were recommended as follow-on developments were: 1) define arbitrary surfaces, 2) determine space-curve intersections of two surfaces, 3) determine edges hidden by arbitrary surfaces, and 4) satisfy general graphical constraints.

3. Subroutine Packages

A popular approach in the early use of graphic systems was to provide subroutine packages callable from higher-level languages. The advantage of these packages is that they allow the user to define his own analysis programs in a language in which he is familiar and at the same time utilize the graphics capabilities. The most widely used packages are GSP [Ref. 6], IGS [Ref. 7], and Sparta [Ref. 8]. The typical form of a subroutine implemented graphic function is CALL name (argument 1, argument 2,..., argument n). The primary difficulty with the subroutine form is the cumbersome use of parameter lists when the number of arguments is large.

GSP was originally designed to be used with FORTRAN IV on the premise that graphic systems would be used for engineering and scientific applications, and that FORTRAN (at that time) was the most widely accepted programming language in these fields. It was decided to use subroutines in lieu of extensions to the FORTRAN language so that the graphic capabilities could also be used within an assembly language program. Additionally, it was decided that GSP should provide only general-purpose graphic capabilities. That is, routines should neither generate geometric figures nor impose constraints on the structure of a model since these tend to be too dependent on application. As a result, elemental image generation is limited to points, lines and characters. More complex images are constructed using these basic elements. This sometimes involves repetitious use of the basic routines.

GSP has reduced some of the problem of long argument lists through the use of default values and null variables. A null variable can be used as a calling argument in which case it merely takes up space

in the sequence of parameters. The value of a corresponding formal parameter in the subroutine is either the last value it was assigned or a default value.

The basic mechanism that permits interaction between the program and the display is attention signals (i.e., status of light pen, function switches, etc.). These signals are interrogated at appropriate intervals during execution of the program. The program, itself, is not interrupted. Hence all signals from the graphic subsystem must be anticipated.

A final feature of GSP which is of interest is that of a replacement buffer (in the main computer). This facilitates animation by having the capability to fill the buffer while the one at the console is being displayed. When the replacement buffer is full, it replaces the buffer in the display console.

IGS (Integrated Graphics System), developed by the RAND Corporation, is a set of subroutines which can be used with languages such as FORTRAN, PL/1 or any other language which utilize Operating System/360 linkage formats. The display terminal utilized is an IBM 2250. One feature in IGS, which reduces the number of parameters passed in subroutine calls, is a special parameter array called "MODE SET ARRAY". This array would contain default values and or values since the last time it was specifically modified. This remains in effect until the next time it is modified or reset. For example, in order to display a character string, all that needs to be specified are the characters themselves, the x/y position of the first character and the number of characters. Other parameters such as character size, line spacing, margins, line orientation, etc. are automatically provided for as default values in the "MODE SET ARRAY".

Although IGS relieves the user from intricate hardware considerations, the functions utilized in IGS are primitive, and hence a long and complex set of calls are required to construct even simple displays.

The user controls the action of the programmed application through the use of the on-line typewriter, function switches, light pens; or by drawing on the RAND Tablet; or by touching user-defined sensitive areas with the light pen or tablet stylus. As in GSP, input from these devices must be specifically planned for.

Sparta, implemented as a package of FORTRAN subroutines, is a procedure oriented programming language for the manipulation of arbitrary line drawings. It was originally designed for use with a CALCOMP plotter, but another version was implemented using an IBM 2250 display. The functions in Sparta are mostly output oriented. But even so, its capability in the area of graphic display generation and manipulation far exceed those in GSP or IGS. The language could be extended to utilize the input capabilities of the graphic devices.

In the area of elemental figure generation, not only are there functions for lines and points, but also rectangles, regular polygons, triangles, stars, ellipses, spirals, etc. Grids can be cartesian, logarithmic or polar. Images can also be input from cards using a read routine which reads one card (record) for each point. Even a general mathematical figure generator which drives any function subprogram written by the user specifying the equation of the desired curve or family of curves is provided.

There are translation routines for "MOVE" and "SIZE". A "NORMAL" function centers a picture and scales it so that it will just

fit inside a display frame without distortion while "FILL" causes a picture to touch the frame on all sides (usually resulting in distortion). Orientation routines provide for rotation, mirror image and oblique transformation (lean).

A general mathematical transformation routine is provided which allows any function subprogram written by the user to be applied to the rectangular or polar coordinates of each point of an image.

There is also a set of random transformations provided to rotate, change size, and move a picture by random amounts within limits in any desired combination. These transformations were included because of an interest in "controlled randomness" studies, random distribution and distortion of pictures, irregularly spaced cross-hatching and surfaces generated by random distribution of points.

The programmer may mix the subroutines of Sparta with any other FORTRAN IV statements.

A major advantage of having Sparta implemented for both plotter and CRT output is in the area of computer-aided design. For instance, an engineer could design an electronic circuit using the CRT graphics capability and when the design is firm, a hard copy could be output to the plotter.

The program and resultant picture in Figure 6 demonstrates the power of the package.

4. Extensions to FORTRAN, PL/1 etc.

GRAF (Graphic additions to FORTRAN) [Ref. 9] provided on additional set of statements within the FORTRAN language to implement on-line graphic capabilities.

```
REAL MAPLE(3,50), EXPO(3,100),
WA(3,1000), PICT(3,1000)
CALL SETMAX(MAPLE,50)
CALL SETMAX(EXPO,100)
CALL SETMAX(WA,1000)
CALL SETMAX(PICT,1000)
CALL OPEN(16.,10.)          Frame 16 by 10
                            inches
CALL READN(MAPLE)           Read and nor-
                            malize maple leaf
CALL READN(EXPO)            Read and nor-
                            malize expo
                            symbol
CALL SIZE(.1,.1,MAPLE,WA)
                            One tenth maple
                            to WA
CALL SIZE(.1,.1,EXPO,PICT)
CALL MVNEXT(WA,PICT,WA,O)
                            Move maple (in
                            WA) to right of
                            expo
CALL CAT(PICT,WA)           Place maple and
                            expo together
                            in PICT
CALL SIZE(.5,.5,MAPLE,WA)
                            Half size nor-
                            malized maple
CALL MOVEBY(-2.,0.,WA,WA)
                            Move 2 inches
                            to left
CALL OVRLAP(PICT,WA,PICT)
                            Move little maple
                            and expo
                            into maple
CALL CAT(PICT,WA)           Combine the three
                            into PICT
CALL OBLIQE(45.,PICT,PICT)
                            Oblique trans-
                            formation on
                            result
CALL DRAWF(PICT)            Draw frame and
                            result

CALL CLOSE
STOP
END
```

FIGURE 6--EXAMPLE OF A PROGRAM IN SPARTA

(Figure 6 was extracted from Ref. 8)

The working force central to GRAF is that of a DISPLAY variable. The value of a display variable is a set of graphic device orders capable of generating a display of points, lines, and characters when sent to the display device. They are declared and can be dimensioned. They may appear in EQUIVALENCE and COMMON statements and passed as subroutine arguments. They can be assigned values in an assignment statement.

An example of a display declaration is

```
DISPLAY A,J,Q(17),TR(2,7,5)
```

where A and J are to be display variables, and Q and TR to be arrays of display variables.

A display expression in GRAF is a sequence of display variables and display functions separated by plus signs. The values of the display variables and display functions are concatenated from left to right to form a string of graphic orders; this string becomes the value of the display expression.

The value of a display function is a string of graphic orders. The built-in display functions of GRAF are:

POINT(X,Y) - Generate orders for plotting a point;

LINE(X,Y) - Generate orders for plotting a line;

PLACE(X,Y) - Generate orders to change beam position without plotting;

CHAR(string, length, mode) - Generate orders for plotting a string of characters; and

PRINT n, list - Create a string of characters using a list and a format almost exactly like the PRINT of FORTRAN and then generate the orders necessary to plot the resulting string of characters.

A display assignment statement is used to assign the value of a display expression to a display variable. For example:

K99=PLACE(0,0)+PRINT 14,(ZK(I),I=1,7)+PLACE(2000,2000)

Display variables can be reset to empty strings with the use of "RESET" subroutine.

When coordinate information is contained in display orders, subroutines are brought into effect to make the transformation from user coordinates to device coordinates.

Control of display variables in relation to the graphic device are handled by FORTRAN functions "PLOT", "UNPLOT", "ERASE" and subroutine "BLANK". "PLOT" transmits the list of display variables to the display device. "UNPLOT" removes the current instance of a display variable and makes the previous current instance the current instance. "ERASE" is the same as "UNPLOT" except all instances of designated display variables are removed. "BLANK" clears the display.

Alphanumeric information is read from the display buffer into main storage by a "READ" statement (works similar to a FORTRAN read).

Cursors are set or removed from a display variable via SETCUR and RMVCUR, respectively.

In order to handle attentions, GRAF must request such information. Interrupt attentions could not be programmed because of operating system constraints under which GRAF was implemented. A "DETECT" function returns information about attentions generated by the light pen, function switches, etc. "DETAIN" is a similar function except that it waits until a specified attention occurs. "DETECT" or "DETAIN" return with a value indicating the type of attention. If the attention was light pen, then the "LPNAME" function can be used to determine which of

its arguments (if any) was the name of the object being pointed to by the light pen.

GRAF also provides the capability for a user to create his own display functions using GRAF statements. For example, a user could create a function called BOX (below) and use it in a manner similar to the built-in functions.

```
DISPLAY FUNCTION BOX(X,Y,U,V)
BOX=PLACE(X,Y)+LINE(U,Y)+LINE(U,V)+LINE(X,Y)
RETURN
END
```

The authors cited that the advantages of this method of implementation are that it is easier to implement, easier to teach to FORTRAN programmers, and easier to read. Also by avoiding the use of call statements with long parameter lists, coding, debugging and understanding the logic of a program are made much easier.

The RAND Corporation developed an extension to a conversational subset of PL/1 which would facilitate the use of computer graphics [Ref. 10]. An IBM Conversational Programming System (CPS) was installed at RAND to provide multi-user on-line access to computer facilities. CPS is entirely typewriter-oriented, and uses a major subset of the PL/1 programming language. An investigation was conducted to determine if the PL/1 of CPS could be expanded to give video graphic CPS users access to all the graphics facilities of the POGO system (see Section 6).

It was found that relatively few language extensions were necessary in order to provide a significant graphics capability within interactive PL/1. The extensions consist of:

1) One new statement, DISPLAY, that signals that the next action will be the creation of a named display page;

2) Additional options in the PUT and GET statements;

3) An additional ON-condition, PUSH, that relates light-pen or stylus actions to asynchronous program responses.

The DISPLAY statement provides the capability for a user to enter a construction mode in which such facilities as those offered by the RAND POGO system are made available to him. A display page so constructed can be referenced as a unit by a label given to it.

The standard PL/1 PUT and GET statements allow data to be placed in or received from a string or a file. In order to incorporate a graphics communication capability within PL/1, all that was necessary was to provide a display page as an additional source or recipient of data.

Finally, ON-conditions in PL/1 provide the capability to activate designated procedures when interrupts occur. It was natural, then, to include the light pen as one of the interrupt sources.

5. Meta-Language Systems

Kulsrud [Ref. 11] proposed that a General Purpose Graphical Language (GPGL) be developed which is device independent and can handle both generation and recognition. Kulsrud further proposed that the GPGL compiler be constructed through the use of a meta-compiler in order that it may be implemented on various hardware configurations. The language described included statements necessary for picture analysis, but does not, however, provide for three-dimensional displays, or more than three levels of hierarchy. The design provides for use of the language in conjunction with other higher-level languages such as FORTRAN, MAD, etc.

GEMS (Graphical Experimental Meta System) [Ref. 12] developed at the Stanford Linear Accelerator Center is a system which facilitates economical experimentation with graphical systems with a linguistic base and provides device independence. A graphical system defined utilizing GEMS can function interactively or in slave mode. Also, the capability exists to create a system which allows for recognition and/or generation of pictures. A graphical system is implemented by defining its components utilizing a simple precedence translator writing system. GEMS is implemented on an IBM 360/91 in PL/1 as three language preprocessors using SIMPLE [Ref. 13] and a comprehensive procedure library for accessing data structures. Typically, a control description is translated into a control processor which acts as the executive program for an application and is essentially the highest level program for the interactive application. A data description is translated into a definition processor. Finally, the graphic description is translated into a parser and associated semantic processors.

6. Specialized Systems

In the area of picture generation and recognition, an interesting concept developed at the Stanford Linear Accelerator Center is that of a Picture Description Language (PDL) and Picture Calculus [Ref. 14]. A primitive in the language is defined as any object with a head and a tail. A primitive class is given a name and a specification of its head and tail, and is defined further by a list of attributes. The attributes contain information on how to generate the desired picture primitive image. The general form is

value=(name, tail, head, attribute 1,...,attribute n).

An example of a primitive for an Arc is arc=(ARC, initial angle, final

angle, curvature, subtended angle). In this case the head and tail information is implicit in the attributes. With primitives as the basic building blocks, concatenation operators are used to specify how pictures are composed from these basic blocks and other concatenated elements.

Given that A represents t——→h and B represents t⌒h, then the use of the concatenation operations can be illustrated as follows:

A + B = t——→h⌒ (head to tail)

A x B = t⌒h (tail to tail)

A - B = t——→h (head to head)

A * B = t⌒h (tail to tail and head to head)

B = h⌒t (tail head reversal)

B = t h (blanking operator)

A final unary operator would have the general form T(w) and provide for linear and non-linear transformation of primitives or pictures. Examples of linear transformations might be move and rotate.

With primitives and operators defined in this way, a string description can be converted into its graphical form (generation) or a graphical form can be converted into string form (recognition) For example, if A = t↓h , B= t——→h , C = h↗t , and D = t↘h then

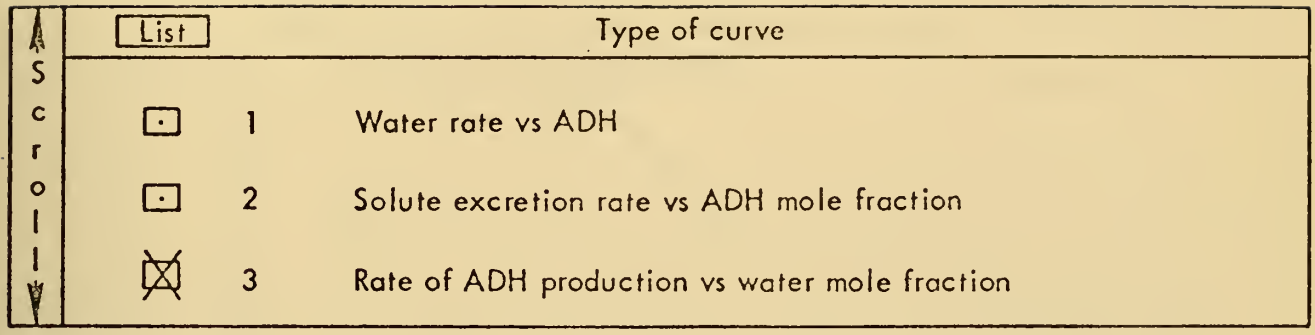
$= A + B + A * B (C + D)$

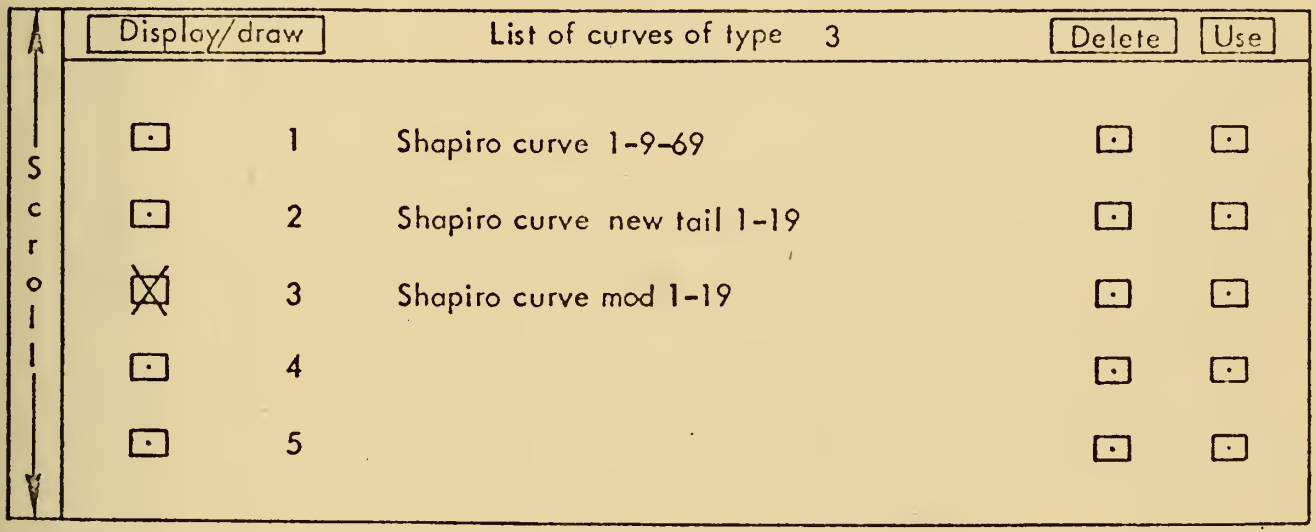
One rule which conserves storage is that a string description is always retained and the vector representation conditionally retained if sufficient storage is available. One problem with the system is that only two attachment points are defined and when two or more objects are concatenated the result only contains a single head and a single tail (two attachment points). This represents a problem when multiple attachment points are required. For example, a resistor in an electronic circuit diagram may have a center tap.

POGO (Programmer-Oriented Graphics Operation) [Ref. 15] is an operational program, written in IGS [Ref. 7] , suited to the interplay of analysis programs with alphanumeric and curve input and display. Except for curve input and display, the system is not designed to dynamically manipulate geometric images. The POGO system provides the capability to 1) design control pages at the graphics terminal and interface the result with a FORTRAN program, 2) trace curves via the RAND tablet into specified FORTRAN arrays, and 3) output curves from specified arrays to the display.

Control pages in POGO are designed by the use of a set of routines which allow the creation of strings of test, fields for numerical values, option boxes, and geometric figures, which can be manipulated by a user. At the press of a button, a set of cards are punched out to be included in the FORTRAN program to recreate the display.

In order to input curves, control pages such as those in Figures 7 and 8 can be created and used. Figure 7 shows a control page that is used to select the category of curve input. By selecting a member of





Page back

Page ahead

FIGURE 7--INPUT CURVE LIST PAGE

(Figure 7 was extracted from Ref. 15)

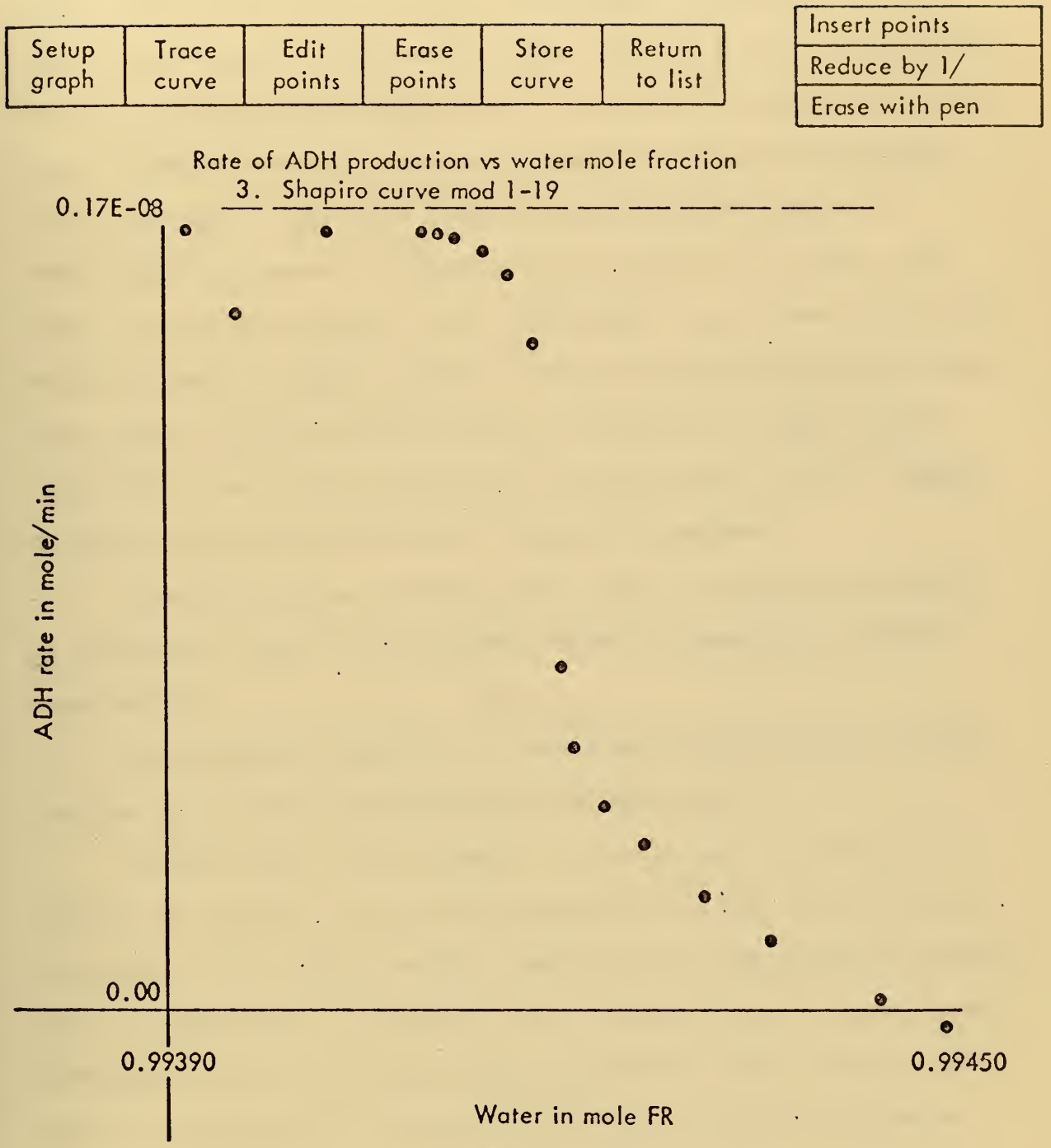


FIGURE 8--POGO CURVE-TRACING PAGE

(Figure 8 was extracted from Ref. 15)

"LIST", a list of curves for that type is displayed. "USE" causes the listed curve to be used in the model and "DELETE" causes it to be removed from the model. The "SCROLL" option allows a user to reach items on the list not currently displayed by "scrolling" the entire list up or down. "DISPLAY" causes a transfer of control to a curve tracing page set up for input of the curve characteristics selected (Figure 8). "TRACE CURVE" is pressed (selected with the light pen) to cause curve input from the RAND tablet. Then, "SETUP GRAPH" can be used to move axis and adjust scale to match the curve. Selected points can be removed with "EDIT POINTS" and the whole curve can be removed with "ERASE POINTS". "STORE CURVE" causes the curve data to be written onto a disk. "RETURN TO LIST" causes the control page of Figure 7 to reappear.

Finally, by using another control page, the curves created can be labeled and linked to the FORTRAN program and appropriate FORTRAN cards punched.

Curve output (display) is accomplished through the use of POGO routines and control pages within the FORTRAN program.

POGO is done in two phases of 1) design and 2) execution and requires two sessions. One of the conclusions was that these two phases be combined into a single session. Additionally, other areas recommended for improvement were: 1) provide a wider range of graphic pages (charts, histograms, etc.); 2) provide simple operation on curves (partial integration, derivatives); 3) provide capability to reformat displays at execution time.

7. Systems Developed at the Naval Postgraduate School

On-line Graphical System [Ref. 16] was an early attempt to implement on-line display in the general purpose hybrid simulation laboratory (XDS 9300 computer, CI5000 analog computer and a TASKER graphical

display system) at the Naval Postgraduate School. For versatility, several low level subroutines were implemented rather than one single processor subprogram. The routines were written in XDS9300 assembly language and callable from FORTRAN IV. In some cases, long calling sequences are required. All data generated by the software package was in addition to the user's data; hence, the original data was not disturbed. The system was capable of displaying vectors, points, and alphanumeric data. Additionally, curves could be drawn, changed and deleted. Plotting of data could be presented in either cartesian coordinates or polar coordinates. The display unit software package contained two major functional groups of 1) curve plotting and 2) tactical situation plotting.

SCOPE [Ref. 17] is an introductory graphics language implemented on an XDS 9300 computer interfaced with an ADAGE AGT 10 graphics subsystem. The language is designed to provide an introduction to interactive computing utilizing the graphics subsystem for input/output. A user can create, display and manipulate a figure in two or three dimensions. Provisions are made in a three dimensional display to define hidden parts of the image for various orientations. The instructions in the language are command oriented and can be input through the card reader or the keyboard attached to the AGT 10. Provisions are also made to correct instruction errors at the AGT 10.

General Purpose Graphic Language (GPGL) [Ref. 18] is an interactive language which is intended for two-dimensional and three-dimensional displays. Popular general purpose graphic languages were compared and a survey of the attributes and capabilities of an interactive general purpose graphic language was documented. The final result was the

definition of GPGL. A subset of GPGL was implemented on the Naval Postgraduate School's ADAGE AGT-10 graphic terminal in a stand alone mode. The primary purpose of the implementation was to demonstrate and evaluate a tri-level hierarchy data structure within the graphical display system. The design provides the capability to define user functions (even of constructing such functions on-line) in addition to the normal system functions. Many of the functions (system or user) are command oriented and can be exercised from the keyboard. The language design also provides functions which interpret topology and other pictorial features. For example, "WITHIN" determines if one component of a picture lies within the second component, and "SEPAR" determines whether two selected images are separate or connected. In the three-dimensional mode, four quadrants are displayed for an image (three views plus a perspective). Any of the quadrants can be designated to occupy the whole screen and at some later time the screen can be returned to the four quadrant mode.

III. GRAPHIC LANGUAGE EXTENSIONS TO FORTRAN

A. DEFINING THE FEATURES OF A GRAPHIC LANGUAGE

In the preceding section, certain specific languages were presented. Other efforts in the area of computer graphic language development have concentrated on defining the attributes and capabilities of general purpose graphic languages; again, the literature is extensive. A comparison of various view points is presented in Table I. Literature references concerned with this area are listed across the top and some of the more widely discussed attributes and capabilities listed down the side.

A brief description of each of the attributes is given below:

(1) Coordinate Specification - A method of relating problem units to display units (e.g. cartesian, polar, logarithmic; scale of x; scale of y, etc.)

(2) Primitive Definition - Many applications involve a set of basic graphic symbols used as picture building elements (e.g. transistors, resistors, diodes, etc. in circuit design).

(3) Picture Construction - Creation of a picture model using primitive elements and other image generation functions. Included in the construction of a picture is the capability to connect picture elements together. Thus levels of hierarchy can be established.

(4) Copy - Storage and time can be saved by copying another image or primitive. A direct copy may be made, or a pointer to the original created.

(5) Constraint Checking - (e.g. Is the line tangent to the circle).

		C. I. Johnson [Ref. 3] 1	S. H. Chasen [Ref. 19] 2	H. B. Baskin S. P. Morse [Ref. 20] 3	E. H. Sibley [Ref. 21] 4	H. B. Baskin [Ref. 22] 5	H. E. Kulsrud [Ref. 11] 6	R. Anderson et al [Ref. 23] 7
1.	Coordinate Specification		X			X	X	X
2.	Primitive Definition	X	X	X	X	X	X	X
3.	Picture Construction (Hierarchy) (Modeling)	X	X	X	X	X	X	X
4.	Copy				X		X	
5.	Constraint Checking				X		X	
6.	Constraint Imposing		X		X			
7.	Topological Analysis	X			X	X	X	
8.	Attention Handling	X	X	X	X			X
9.	Display Input/Output	X	X	X	X	X	X	X
10.	Picture (Image) Transformation	X	X	X	X	X	X	X
11.	Alphanumeric Text	X	X			X		X
12.	Erase/Delete		X	X			X	X
13.	Store/Retrieve					X	X	
14.	Menu				X			X
15.	Hard Copy		X				X	
16.	Color Coding							X

TABLE I - GRAPHIC LANGUAGE ATTRIBUTES

(6) Constraint Imposing - (e.g. Construct a perpendicular line segment from point A to the line CD).

(7) Topological Analysis - (e.g. Does image A intersect image B).

(8) Attention Handling - Routines to handle such devices as light pen, function switches, joystick, etc. either by interrupt or by polling.

(9) Display Input/Output - The input or output of text and graphics between the computer and the graphics display.

(10) Picture (Image) Transformation - Translate, rotate, etc.

(11) Alphanumeric Text - Capability to display text.

(12) Erase/Delete - Erase means that a specified image is removed from view and delete causes the image to be purged from storage.

(13) Store/Retrieve - Capability to store information onto an auxiliary storage device and to later retrieve it.

(14) Menu - The capability to display some primitive images on the screen and to select one of these for a copy to be used in constructing a picture, or to display program control options and select from them at the console.

(15) Hard Copy - Capability to output designated images onto a line printer or X-Y plotter.

(16) Color Coding - Capability to specify the color of a displayed image.

Baskin and Morse [Ref. 20] describe an approach to graphic language design in which functions would be partitioned into application-independent and application-dependent subsets. The application-independent functions are referred to as conversational functions and provide the user with a simple language for describing a pictorial representation of his problem to the computer. Application-dependent functions are non-graphic and analytical in nature and are oriented to the particular application.

Johnson [Ref. 3] stated that the programming of graphic display systems is difficult unless easy-to-use support routines are provided. Preferably, the routines should be implemented in or accessible from a high-level language.

B. GRAFTRAN (GRAPHIC EXTENSIONS TO FORTRAN)

FORTRAN is a widely used language for analytical problems. By using FORTRAN as a vehicle for implementing graphic capabilities, the accessibility of graphic support routines from a high-level language is satisfied, as suggested by Johnson, and the two level partitioning described by Baskin and Morse is provided. FORTRAN would provide the analytical partition and the graphics extensions would provide the conversational partition.

The user of a general purpose computer graphics system should have available a wide variety of operations to aid him in the solution of his problem. The operations available in the language should be apparent so that the user can devote the majority of his attention to the application. The authors consider the following features to be a baseline of operations which will allow the user to easily use a computer graphics system in the solution of his particular problem.

A user should not be required to perform his own data scaling and axis layout. Hence, a Cartesian coordinate specification procedure should be provided which relates the user data to the scope coordinates and specifies the scale of user units to scope inches. Likewise, it should be possible to describe basic primitive images (e.g. symbols in a flowchart) and connecting information to construct higher level images. This form of primitive definition and hierarchial image construction should be provided.

To aid the user in image construction, a copy function saves time (it also saves storage in that common information can be pointed to from various images and primitives of the same type). It is also desirable to label images and display other textual information on the graphic console. Thus, text naming and display features should be provided. As an initial base-line capability, a display operation should cause text and graphic entities to be displayed on the graphics console. Finally, operations should be provided to poll the function switches, joystick, variable control dials and pedals.

In Appendix B, a graphic language implementation in FORTRAN, called GRAFTRAN (Graphic Extensions to FORTRAN) is described. GRAFTRAN is implemented as a working base-line system which can be readily expanded.

IV. GRAFTRAN IMPLEMENTATION ON XDS9300/AGT10

A. PREPROCESSOR

There are basically two methods that can be used to create a new language: one can either write a new compiler or modify an existing compiler. Since GRAFTRAN was being implemented on an experimental basis and the time available was insufficient to write an entirely new compiler, an augmentation of the existing compiler was undertaken. One particular method of augmentation which has several desirable features is a preprocessor; GRAFTRAN is implemented in this manner. A complete description of the preprocessor can be found in Appendix D.

One reason that the preprocessor method was chosen is that it allowed using the present compiler, written in assembly language, without having to modify it. Another attractive feature of a preprocessor is that it allows changes to be easily made.

SDS FORTRAN IV [Ref. 24] and assembly language [Ref. 25] are the only languages available at the Naval Postgraduate School (N.P.S.) Computer Laboratory. The preprocessor for GRAFTRAN was written in SDS FORTRAN IV and was implemented on an XDS 9300 computer which is described in Appendix A. One helpful feature supplied by FORTRAN is the debugging facilities.

One other important consideration prompted the use of FORTRAN and the preprocessor method. Since there are several additions which could be made to GRAFTRAN, (see Conclusion and Future Developments) it was felt that individuals interested in this subject could modify the preprocessor more easily if it were written in FORTRAN rather than assembly language.

B. SELECTION OF A GRAPHIC DATA STRUCTURE

To have an effective interactive graphic system, there are several necessary attributes which must apply to the data structure. It is imperative that the data structure present and accept data quickly in order to maintain an efficient interplay between man and machine. It also must be flexible enough to allow updating so the user can interact with the computer to influence the course of his computations. Another useful tool in many applications is a hierarchial structure. This attribute allows the user to manipulate an entire structure or subordinate elements within a given structure. Another helpful characteristic is the ability to easily use the data structure. Too often a user is forced to concentrate on programming to produce the data structure, rather than focusing on the problem to be solved. The following paragraphs describe several data structures which were considered as candidates to be used in conjunction with GRAFTRAN.

There are several data structures which lend themselves well to interactive graphics or which have been developed expressly for that purpose. Some of the attributes of the structures to be discussed are summarized on Table II; a short description of each structure will follow. All of these structures are list structures where the elements (or records) are chained together by pointers. An obvious disadvantage of this type of structure is that it consumes more storage than a random or sequential structure. However the advantages of easily inserting, deleting, and updating records (or entire files) far outweigh the disadvantages. This type of structure also lends itself easily to a hierarchial mechanism.

	L6	LEAP	ASP	APL	SLIP
HIGHER LEVEL LANGUAGE	NO	YES (1)	NO	YES (2)	YES
HIERARCHIAL MECHANISM	YES	YES	YES	YES	YES
GENERAL APPLICATIONS	YES	NO (4)	YES	YES	YES
INDEPENDENT FROM USER	NO	YES	NO	YES	NO (3)
PAGING SYSTEM	NO	YES	--	YES	YES
MACHINE INDEPENDENT	NO	--	NO	YES	YES
DEBUGGING AIDS	YES	NO	YES	--	YES

1. ALGOL

2. PL1

3. FORTRAN PLUS A FEW MACHINE DEPENDENT PRIMITIVE ROUTINES.

4. ASSOCIATIVE TRIPLES

TABLE II - DATA STRUCTURE ATTRIBUTES

Bell Telephone Laboratories' Low-Level Linked List Language (L6) [Ref. 26] was developed expressly for list structure manipulations. It allows the programmer to build a wide variety of linked data structures composed of variable size blocks - a sequential number of machine words, and specify fields within a block. Blocks may be interconnected by pointers which effectively can define a ring or tree structure in accordance with a users needs. The Author states "L6 is useful where the programmer wants the pattern of pointers among data items to correspond closely with which his program deals" [Ref. 26] . He also implies that programs will be fast-running since the program is written in machine code. Although no figures were available and since it is written in machine code, it seems logical that both core requirements and speed would largely depend on the characteristics of the computer involved, and the users ability to write an efficient program.

The LEAP language and its data structure [Ref. 27] was developed to express data relations in associative structures. Data associations are in the form: "Attribute of Object is Value", or in set notation, A(O)=V, where all sets are organized in a ring structure. A paging system is mandatory since attributes, objects, and values are stored on separate pages. This necessitates considerable duplication of data; thus, storage requirements are vastly increased - all but the smallest structures require secondary storage [Ref. 28] . A hash coding technique is used to create and retrieve data. Retrieval of the data can be any of the following forms: (1) Given A, O, and V, the structure can be searched by hashing A, O, and V, to see if the triple is in the structure, (2) Given A and O, or A and V, or O and V, the two given items are hashed to locate the third item; and (3) Given any one item, all data

about that item is returned. It should be noted that the structure is applicable only when the data is already in the form of A(0)=V. For this reason the usefulness of this structure is questionable for general purpose applications.

The Associated Structures Package (ASP) [Ref. 29] is a general purpose data structure which allows a user to create and manipulate ring structures. The user may also define an arbitrary number of relationships between data items. The following actions can be performed using ASP: (1) store, access, and delete arbitrary data items; (2) create, manipulate, and delete complex relationships between items; (3) manage dynamic storage. One of the most interesting features of this language is the ability to compile statements in ASP together with programs written in other languages. This allows a programmer to build a model in ASP and perform calculations on the same model with another language more suitable for analysis. In this manner, efficiency for graphics and for analysis can be effectively achieved.

Associative Programming Language (APL) [Ref. 30] offers facilities for declaring relationships between data elements and restructuring the data structures as the program progresses. Since it is designed to be embedded in PL/1, it contains all the advantages of PL/1 plus (1) symbolic data references, (2) hierarchy mechanism, (3) n-dimensional data association, and (4) automatic extension of addressable core beyond the confines of high speed core. APL can operate in both a paging or non-paging mode; however, in the non-paging mode, addresses of all data items are absolute and space is limited to core storage.

Weizenbaum's Symmetric List Processor (SLIP) [Ref. 31] is described in Appendix C. As Table II indicates, it possesses all the attributes

necessary for an interactive graphics system. It has been suggested [Ref. 32] that SLIP is compatible with a medium sized computer such as the XDS9300. Since SLIP was programmed in FORTRAN and already implemented Ref. 32 on the XDS9300, it was chosen as the structural basis of GRAFTRAN. The GRAFTRAN preprocessor will output the necessary calls to SLIP to create any model a programmer desires. The user may also use any of the SLIP subroutines needed to create a structure tailored to his needs.

V. CONCLUSIONS AND RECOMMENDED FUTURE DEVELOPMENTS

Several graphics languages exist that allow extremely complex operations. Unfortunately, these languages are usually difficult to use and require a large amount of memory. The objectives of GRAFTRAN as discussed in Section III. B., enable the user to easily create and display a primitive, image, or text on the graphics console of a medium scale computer graphics system.

A. OPERATIONAL SYSTEM DESCRIPTION

GRAFTRAN offers a broad base-line of operations which can be performed on primitives, images, and text. To display an element, the user simply declares, defines, and displays it. The user is completely isolated from the specifics of manipulating the data structure and the calls for the interface routines which are used to communicate between the computer and the graphics console.

As explained in Appendix B, images are comprised of primitives or other images; this allows an extensive hierarchial system. The method employed in the hierarchial system is the use of attachment point pairs. Attachment points, which are declared in the definition of an image, indicate which attachment points of images or primitives are to be joined. The depth of the hierarchical level that can be obtained is restricted only by the limits of available memory of the computer system being used. The mechanism to connect the attachment points in order to display the image is again completely isolated from the user.

Since core memory is a prime consideration for all programs on a small or medium scale computer system, GRAFTRAN possesses a copy feature.

This allows multiple copies of a primitive or image which contain pointers to the data arrays which define the primitives. Since the data arrays do not have to be reproduced, the core memory saved is a function of the number of copies of a primitive (or image) multiplied by the size of the data arrays involved. This is especially useful where several primitives (or images) are repeated in a structure such as a transistor element in circuit design.

The Cartesian center and scale factor used in a structure are dependent on particular program applications. GRAFTRAN allows a user to declare any two-dimensional Cartesian coordinate system. The user may also declare a scale factor which is interpreted as units per scope inch. This eliminates the burden of scaling every structure the user desires to display.

Several operations can be applied to primitives. The intensity of a primitive has a default value of maximum intensity when it is defined. The user, however, can reset the intensity to any value. Similarly the primitive has default values of a solid vice dash mode and draw vice move mode. These values may be also selectively redefined. This is useful in displaying special effects. For example, a hidden line in an image could be displayed with less intensity and in the dash mode to produce the desired effect.

Occasionally it is necessary or desirable to rotate an image on the graphics console. If the image is complex, this is often a time consuming and laborious process; not so with GRAFTRAN. The user simply indicates the amount of rotation and the name of the image or primitive that is to be rotated. The entire image, regardless of the level of hierarchy, will be rotated the indicated amount.

A translation operation functions similarly as the rotation operator. All translations will be automatically calculated from the origin of the declared Cartesian coordinate system. This naturally allows the user the flexibility of displaying disjoint structures in relation to each other or for comparison purposes.

GRAFTRAN, in its implementation at the N.P.S. Computer Laboratory, is relatively fast in displaying structures on the graphics console. It appeared to take approximately two seconds to display ten primitives which had been translated and/or rotated in a three level hierarchial structure. The Authors consider that this is suitable for interactive graphics application.

B. RECOMMENDATIONS FOR FUTURE DEVELOPMENT

As mentioned previously, GRAFTRAN offers a broad base-line of operations. In its present form, complex structures can be manipulated and displayed. The ability exists to control the flow of a program at the graphics console with the use of all the attention devices except the light pen. Future development of GRAFTRAN should include expanding the interaction capabilities. Appendix G gives a complete summary of features which should be considered for future development of GRAFTRAN.

APPENDIX A COMPUTER LABORATORY

The Computer Laboratory at the Naval Postgraduate School (N.P.S.) consists of an XDS9300 digital computer system complemented by a CI-5000 analog computer system and two AGT/10 graphics computer systems. These systems may be used singly or in combination with one or more of the other systems.

The XDS9300 is the main computer with 32k words (24 bits) of main memory and a magnetic drum for secondary storage. Input to the XDS9300 can be made via card reader, paper tape reader, magnetic tape, or tele-typewriter, while output can be made via line printer, paper tape punch, tele-typewriter or magnetic tape drive.

The languages available on the N.P.S. XDS9300 computer system include SDS FORTRAN IV [Ref. 24] - STANDARD FORTRAN IV with some additional features, and assembly language [Ref. 25]. An additional feature on the XDS9300 is the ability to intermix FORTRAN IV with assembly language. Since the XDS9300 is a word oriented computer, the ability to use assembly language within FORTRAN proved to be invaluable when the manipulation of selected bits within a word was required.

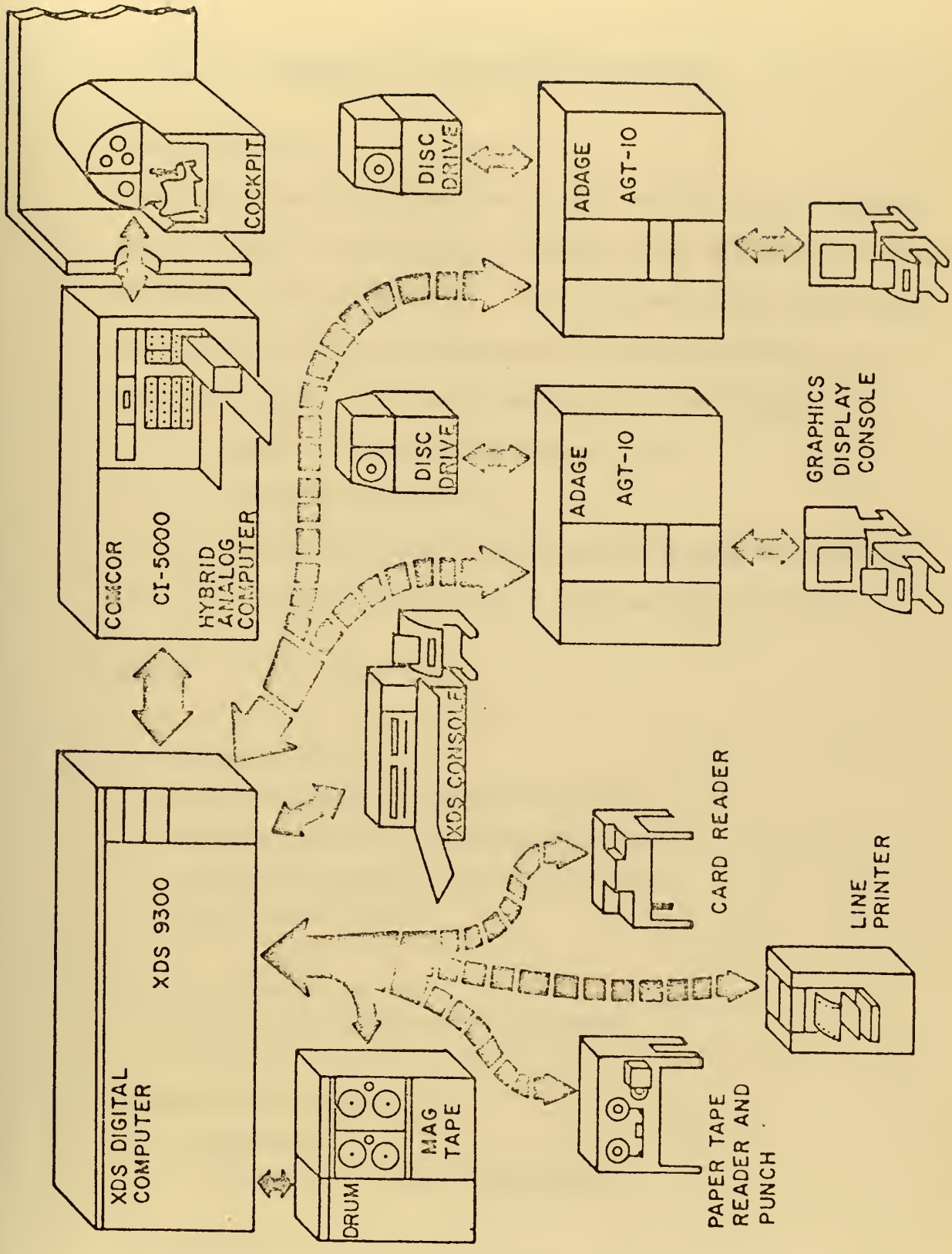


FIGURE 9--N.P.S. COMPUTER LABORATORY CONFIGURATION

(Figure 9 was extracted from Ref. 33)

APPENDIX B DESCRIPTION OF GRAFTRAN

1. GRAFTRAN Program

GRAFTRAN is completely described in this section using a metalinguistic notation first developed by Backus for describing ALGOL in the ALGOL 60 Report [Ref. 35]. The notation is called BNF, which is an abbreviation for Backus-Normal Form (also referred to as Backus-Naur Form). The basic symbols of the metalanguage are:

: :=	connective meaning "is defined to be"
|	connective meaning "or"
< >	delimiting brackets enclosing metalinguistic variables

For a more complete description of this language, see Ref. 4.

Syntax

```
<GRAFTRAN PROGRAM>::=
    <MAIN PROGRAM><SUBPROGRAM BLOCK>
    |<SUBPROGRAM BLOCK><MAIN PROGRAM>
    |<SUBPROGRAM BLOCK><MAIN PROGRAM>
    <SUBPROGRAM BLOCK>

<MAIN PROGRAM>::=
    <STATEMENT BLOCK><END STATEMENT>

<SUBPROGRAM BLOCK>::=
    <SUBPROGRAM>
    |<SUBPROGRAM BLOCK><SUBPROGRAM>

<SUBPROGRAM>::=
    <SUBPROGRAM HEADER STATEMENT>
```

```
    <SUBPROGRAM STATEMENT BLOCK>
    <END STATEMENT>

<SUBPROGRAM STATEMENT BLOCK>::=
    <SUBPROGRAM STATEMENT>
    |<SUBPROGRAM STATEMENT BLOCK>
    <SUBPROGRAM STATEMENT>

<STATEMENT BLOCK>::=
    <STATEMENT>
    |<STATEMENT BLOCK><STATEMENT>

<SUBPROGRAM STATEMENT>::=
    <STATEMENT>
    |<RETURN STATEMENT>

<STATEMENT>::=
    <FORTRAN STATEMENT>
    |<GRAFTRAN STATEMENT>
```

Semantics

GRAFTRAN is an extension of ASA FORTRAN providing the capability to interact with a cathode-ray tube (CRT) graphics unit and to perform certain processes involving graphic and text entities. A GRAFTRAN program consists of a main program plus zero or more subprograms. The main program and each subprogram ends with an "END" statement. Statements within a main program or a subprogram serve to declare types of entities, to perform operations on entities, and to establish control during the running of the program. An implementation of GRAFTRAN using XDS9300 FORTRAN [Ref. 24] is described in Chapter IV. The remainder of this section will describe the features added to FORTRAN to achieve the desired graphics and text capability. The

reader should note that GRAFTRAN is an augumentation of FORTRAN.

Examples:

For an example of a GRAFTRAN program see Appendix F - Sample Run.

2. Basic Symbols, Identifiers, Numbers, Strings

a. Basic Symbol

Syntax

```
<BASIC SYMBOL>::=
   <LETTER>|<DIGIT>|<DELIMITER>

<LETTER>::=
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S
   |T|U|V|W|X|Y|Z

<DIGIT>::=
   0|1|2|3|4|5|6|7|8|9

<DELIMITER>::=
   <SEPARATOR>
   |<VOCABULARY WORD>

<SEPARATOR>::=
   ,|(|)

<VOCABULARY WORD>::=
   TEXT|PRIMITIVE|IMAGE|DISPLAY|XY|MOVE
   |DASH|TRANSLATE|ROTATE|INTENSITY
   |GRAPHICS|LIST|POINT|LINE|ARC|COPY
   |PEDALS|DIALS|SWITCHS|JOY|INPUT|END
   |RETURN|RESET
```

Semantics

GRAFTRAN is composed of a number of basic symbols. The upper-case, Roman alphabet is used to form identifiers and strings. Decimal digits are used to form identifiers, numbers, and strings. Delimiters are combinations of one or more digits, letters or special characters which have fixed meanings within the language and therefore themselves form basic symbols.

b. Identifiers

Syntax

```
<IDENTIFIER>::=
    <LETTER>
    |<IDENTIFIER><DIGIT>
    |<IDENTIFIER><LETTER>
```

Semantics

Identifiers are sequences of from one to 256 letters and digits and must begin with a letter. (In the XDS9300 FORTRAN implementation, only the first 8 positions are used and hence two different identifiers must be unique in the first 8 positions.)

Examples:

TITLE

DATE

TRANSISTOR

CIRCUIT

c. Numbers

Syntax

```
<INTEGER>::=
<UNSIGNED INTEGER>
|+<UNSIGNED INTEGER>
|-<UNSIGNED INTEGER>

<REAL>::=
   <UNSIGNED REAL>
   |+<UNSIGNED REAL>
   |-<UNSIGNED REAL>

<UNSIGNED REAL>::=
   <UNSIGNED INTEGER>
   |<DECIMAL FRACTION>
   |<UNSIGNED INTEGER><DECIMAL FRACTION>

<DECIMAL FRACTION>::=
   .<UNSIGNED INTEGER>

<UNSIGNED INTEGER>::=
   <DIGIT>
   |<UNSIGNED INTEGER><DIGIT>
```

Examples:

0

0.0

0.123

+796

-0.123

512.24

d. Strings

Syntax

```
<STRING>::=
    '<CHARACTERS>'|''

<CHARACTERS>::=
    ANY*BCD*CHARACTER*EXCEPT'
```

Semantics

Strings are sequences of BCD characters other than '. ' is used to delimit a string. (In the XDS9300 FORTRAN implementation of GRAFTRAN, the maximum length of a string is 128 characters).

Examples:

```
''
'XYZ'
'15+23'
'GRAFTRAN PROGRAM'
```

3. Statements

Syntax

```
<FORTRAN STATEMENT>::=
    FORTRAN*STATEMENT

<SUBPROGRAM HEADER STATEMENT>::=
    SUBPROGRAM*HEADER*STATEMENT

<END STATEMENT>::=
    END

<RETURN STATEMENT>::=
    RETURN
```

```
<GRAFTRAN STATEMENT>::=
    <DECLARATION STATEMENT>
    |<COORDINATE SPECIFICATION STATEMENT>
    |<TEXT DEFINITION STATEMENT>
    |<PRIMITIVE DEFINITION STATEMENT>
    |<ATTRIBUTE SELECTION STATEMENT>
    |<IMAGE DEFINITION STATEMENT>
    |<TRANSFORMATION STATEMENT>
    |<RESET STATEMENT>
    |<INPUT STATEMENT>
    |<OUTPUT STATEMENT>
```

Semantics

The terminal symbol "FORTRAN*STATEMENT" refers to any legal FORTRAN statement. The terminal "SUBPROGRAM*HEADER*STATEMENT" refers to the subset of FORTRAN statements which identify a subprogram. "END" is a specific FORTRAN statement which is used to indicate the end of a main program or a subprogram. "RETURN" statement is used by a subprogram to return control to the program which called on it.

Each GRAFTRAN statement occupies columns 7 through 72 of an 80 column record and may be continued on additional records in the same columns by denoting each continuation with a BCD character in column 6. Blanks are ignored between basic symbols. The only restriction is that basic symbols are not to be broken between two records. Note that no provision is made for labeled GRAFTRAN statements. This is not a severe restriction, however, since the same effect can be achieved with the FORTRAN*STATEMENT "CONTINUE". (In the XDS9300 FORTRAN implementation, each GRAFTRAN statement including each continuation record is uniquely identified by the BCD character G in column 1).

Examples:

For examples of GRAFTRAN statements, see the individual statement definitions listed below.

a. Declaration Statement

Syntax

```
<DECLARATION STATEMENT>::=
   <DECLARATION TYPE><IDENTIFIER LIST>

<DECLARATION TYPE>::=
   TEXT |PRIMITIVE |IMAGE

<IDENTIFIER LIST>::=
   <IDENTIFIER>
   |<IDENTIFIER LIST>,<IDENTIFIER>
```

Semantics

A declaration associates, for certain parts of a program, identifiers with certain attributes. Identifiers declared "TEXT" are associated with text strings. "PRIMITIVES" are basic graphic elements upon which "IMAGES" are built. Primitives can have certain attributes such as INTENSITY, DASHED, SOLID etc. Images are collections of other images and primitives connected in a specified fashion.

Examples:

```
TEXT TITLE, SUBTITLE

PRIMITIVE TRANSISTOR, RESISTOR, CAPACITOR

IMAGE CIRCUIT, CIRCUIT 2, MODULE
```

"TITLE" and "SUBTITLE" are defined as names for text strings. "TRANSISTOR", "RESISTOR", and "CAPACITOR" are names of primitive

image forms (i.e. lowest level of hierarchy). On the other hand, "CIRCUIT", "CIRCUIT2", and "MODULE" are defined as image names and hence refer to a connected set of primitives and other images. For instance, "CIRCUIT" might be an image created by connecting "TRANSISTOR", "RESISTOR", and "CAPACITOR" in a specified fashion. "MODULE" might be the connection of "CIRCUIT" and a copy of "CIRCUIT" (e.g. "CIRCUIT2").

b. Coordinate Specification Statement

Snytax

```
<COORDINATE SPECIFICATION STATEMENT>::=
   XY<SPECIFICATION PARAMETER LIST>

<SPECIFICATION PARAMETER LIST>::=
   (<SPECIFICATION PARAMETERS>

<SPECIFICATION PARAMETERS>::=
   <CRT-X><CRT-Y><CRT-SCALE>

<CRT-X>::=
   <REAL>,|,

<CRT-Y>::=
   <REAL>,|,

<CRT-SCALE>::=
   <REAL>)|,
```

Semantics

The purpose of the Coordinate Specification Statement is to establish the relationship between the graphics CRT coordinate system and that of the user program. A Cartesian coordinate

system is established CRT-X and CRT-Y scope inches in X and Y, respectively, from the center of the CRT. The number of user specified problem units per inch in each axis is established by CRT-SCALE. The absence of a parameter means that a default value will be selected.

Examples:

```
XY(-5.0,-5.0,1.0)
```

The center of a Cartesian coordinate system is established 5.0 inches to the left and 5.0 inches down from the center of the CRT. The scale of 1.0 implies that each inch on the CRT represents one user unit (e.g. if a user was using a meter as his problem unit, then the scope would display one meter per inch).

c. Text Definition Statement

Syntax

```
<TEXT DEFINITION STATEMENT>::=
    <TEXT VARIABLE>=<TEXT RIGHT PART>

<TEXT RIGHT PART>::=
    <TEXT ATTRIBUTE LIST>
    |<STRING>
    |<TEXT ATTRIBUTE LIST><STRING>
<TEXT ATTRIBUTE LIST>::=
    <LINE NUMBER><CHARACTER POSITION>
    <SIZE><INTENSITY>

<LINE NUMBER>::=
    <INTEGER>,|,
```

<CHARACTER POSITION>::=
<INTEGER>,|,

<INTENSITY>::=
<INTEGER>)|)

Semantics

The Text Definition Statement is used to assign a string of text to a text variable and to set up the parameters of the string when it is displayed on a graphics console. The attributes consists of 1) line number, 2) character position, 3) size, and 4) intensity of the string when it is displayed on the CRT. Whenever any one of the attribute parameters is left blank in a text definition statement, its setting is left unchanged. When the text variable is declared, all of its attributes are initialized to a set of default values. The default values and the range of numbers acceptable for each attribute depend on the particular graphics equipment utilized.

Examples:

```
TITLE=(20,40,1,3) 'DEMO PROGRAM'
AUTHOR=(35,35,,)
DATE='DECEMBER 3, 1972'
```

The string 'DEMO PROGRAM' is assigned to the text variable "TITLE". When output by a "DISPLAY" statement, "TITLE" is displayed at line number 20 and character position 40. Its size will be 1 and its intensity 3.

The string identified by "AUTHOR" has had its line number and character position changed as indicated, but the size and intensity remain at their previous values (which may still be

default values). "DATE" has been assigned the string 'December 3, 1972'. Its attributes have not changed.

d. Primitive Definition Statement

Syntax

```
<PRIMITIVE DEFINITION STATEMENT>::=
    <IMAGE GENERATOR FUNCTION>
    /<PRIMITIVE ATTACHMENT POINT LIST>/ |COPY
    <PRIMITIVE LIST>

<IMAGE GENERATOR FUNCTION>::=
    <LIST>
    |<POINT>
    |<LINE>
    |<ARC>

<LIST>::=
    LIST(<REAL ARRAY>,<INTEGER>)

<POINT>::=
    POINT

<LINE>::=
    LINE(<SECCND POINT PAIR>)

<SECCND POINT PAIR>::=
    <X>,<Y>

<X>::=
    <REAL>

<Y>::=
    <REAL>

<ARC>::=
    ARC(<ARC PARAMETERS>)
```

```
<ARC PARAMETERS>::=
   <INITIAL ANGLE>,<FINAL ANGLE>,<RADIUS>

<INITIAL ANGLE>::=
   <REAL>

<FINAL ANGLE>::=
   <REAL>

<RADIUS>::=
   <REAL>

<PRIMITIVE ATTACHMENT POINT LIST>::=
   <ATTACHMENT POINT>
   |<PRIMITIVE ATTACHMENT POINT LIST>
   <ATTACHMENT POINT>

<ATTACHMENT POINT>::=
   (<AX>,<AY>)

<AX>::=
   <REAL>

<AY>::=
   <REAL>
```

Semantics

A primitive variable can be assigned a primitive image type and given attachment points through the Primitive Definition Statement. The primitive image is assigned through the use of some basic primitive image generator functions such as line, point, arc, or a user specified coordinate list. Attachment points can be affixed to the primitive so that it can be connected at some later time to other primitives or images.

Examples:

RESISTOR=LIST(RARRAY,15) /(0.0,0.0)(2.0,0.0)/

BUS=LINE(4.0,0.0) /(0.0,0.0)(4.0,0.0)/

"RESISTOR" is defined as a list of points in the array "RARRY" of length 15 points. Its first attachment point is 0.0,0.0 and the second attachment point is 2.0,0.0. "BUS" is defined as a line from 0.0,0.0 to 4.0,0.0 and these points are also defined as the attachment points. All primitives have their own relative coordinate system. That is why the line function has only one x/y pair as an argument. This x/y pair represents the second point of the line and (0.0,0.0) is assumed for the first point. A transformation block is provided with each primitive for translation and rotation of the primitive. Whenever a primitive is a member of another higher level image, the translation information is bypassed since it will be positioned according to attachment specification. The rotation, however,is taken into account. Hence, primitives may be moved about without affecting their position in other images.

e. Attribute Selection Statement

Syntax

```
<ATTRIBUTE SELECTION STATEMENT>::=
    <MOVE STATEMENT>
    |<DRAW STATEMENT>
    |<DASH STATEMENT>
    |<SOLID STATEMENT>
    |<INTENSITY STATEMENT>

<MOVE STATEMENT>::=
    MOVE <PRIMITIVE LIST>
```

```
<DRAW STATEMENT>::=
   DRAW <PRIMITIVE LIST>

<DASH STATEMENT>::=
   DASH <PRIMITIVE LIST>

<SOLID STATEMENT>::=
   SOLID <PRIMITIVE LIST>

<INTENSITY STATEMENT>::=
   INTENSITY(<INTENSITY VALUE>)
   <PRIMITIVE LIST>

<INTENSITY VALUE>::=
   <INTEGER>

<PRIMITIVE LIST>::=
   <PRIMITIVE VARIABLE>
   |<PRIMITIVE LIST><PRIMITIVE VARIABLE>

<PRIMITIVE VARIABLE>::=
   PRIMITIVE*VARIABLE
```

Semantics

Since primitives are the construction elements for higher level images, they are assigned attributes which determine how they are displayed. The line segments of a primitive can be "DASHED" or "SOLID". The intensity of a primitive can be varied. When a primitive is visible on the screen, it has the attribute "DRAW". The primitive can be blanked by the "MOVE" attribute. When primitives are defined by the primitive declaration

statement, they are preset with default attribute values of solid, draw, and intensity at maximum.

Examples:

```
MOVE BOX,CIRCLE

INTENSITY(10) TRANSISTOR, RESISTOR, DIODE

DASH CONNECTOR, FRAME
```

The primitives "BOX" and "CIRCLE" are invisible when displayed on the CRT. The intensity of the primitives "TRANSISTOR", "RESISTOR" and "DIODE" are each reset to 10. The subsequent display of the primitives "CONNECTOR" and "FRAME" will appear dashed.

f. Image Definition Statement

Syntax

```
<IMAGE DEFINITION STATEMENT>::=
    <ATTACHMENT PAIR LIST>
    /<ATTACHMENT POINT LIST>/
    COPY <IMAGE LIST>

<ATTACHMENT PAIR LIST>::=
    <ATTACHMENT PAIR>
    |<ATTACHMENT PAIR LIST>
    <ATTACHMENT PAIR>

<ATTACHMENT PAIR>::=
    (<ATTACHMENT ELEMENT>:
    <ATTACHMENT ELEMENT>)

<ATTACHMENT ELEMENT>::=
    <PRIMITIVE VARIABLE>
    <ATTACHMENT NUMBER>
    |<IMAGE VARIABLE><ATTACHMENT NUMBER>
```

```
<ATTACHMENT NUMBER>::=
    (<INTEGER>)

<ATTACHMENT POINT LIST>::=
    <ATTACHMENT TYPE>
    |<ATTACHMENT POINT LIST>
    <ATTACHMENT TYPE>

<ATTACHMENT TYPE>::=
    <PRIMITIVE VARIABLE>
    <ATTACHMENT NUMBER>
    |<IMAGE VARIABLE><ATTACHMENT NUMBER>
    |<ATTACHMENT POINT>
```

Semantics

Images are constructed from other images and primitives by attaching them together. The resultant image can then be assigned its own attachment points selected from the attachment points of its members. The members of an image are joined using attachment pairs. The lefthand component of the first pair becomes the reference image. Its relative center becomes the relative center of the new image. For each other attachment pair, its lefthand component must have been previously defined in a preceding attachment pair (this insures connectivity of the member images). Attachment points can be defined for the new image by selecting specified attachment points from its member images. The new image is also defined with its own transformation block so that it can be positioned or rotated. As in

primitives, the translate values are bypassed when the image is used to construct a higher level image.

Examples:

```
TRL=(XISTOR(3):CON1(1)) (XISTOR(1):RES1(1))
(XISTOR(2):CON2(2)) /CON2(1),CON1(2),RES1(2)/
```

Assume "XISTOR", "CON1", "CON2", and "RES1" are primitives. The reference member for "TRL" is "XISTOR". All other members are slave to "XISTOR". "TRL" is constructed by attaching the first attachment point of"CON1" to the third attachment point of "XISTOR", the first attachment point of "RES1" to the first attachment point of "XISTOR", and finally, the second attachment point of "CON2" to the second attachment point of "XISTOR". The first attachment point of "TRL" is defined as the first attachment point of "CON2", the second attachment point as the second attachment point of "CON1", and the third attachment point as the second attachment point of "RES1".

g. Transformation Statement

Syntax

```
<TRANSFORMATION STATEMENT>::=
    <ROTATE STATEMENT>
    |<TRANSLATE STATEMENT>

<ROTATE STATEMENT>::=
    ROTATE(<REAL>)<PRIMITIVE/IMAGE LIST>

<TRANSLATE STATEMENT>::=
    TRANSLATE(<TRANSLATE PARAMETERS>
    <PRIMITIVE/IMAGE LIST>

<TRANSLATE PARAMETERS>::=
    <DX><DY>
```

```
<DX>::=
    <REAL>,|,

<DY>::=
    <REAL>)|)

<PRIMITIVE/IMAGE LIST>::=
    <PRIMITIVE VARIABLE>
    |<IMAGE VARIABLE>
    |<PRIMITIVE/IMAGE LIST>
    <PRIMITIVE VARIABLE>
    |<PRIMITIVE/IMAGE LIST>
    <IMAGE VARIABLE>

<IMAGE VARIABLE>::=
    IMAGE*VARIABLE
```

Semantics

Transformation Statements cause images or primitives to be translated and/or rotated.

Examples:

```
ROTATE(-90.0) CON4,RESU
```

"CON4" and "RES2" are each rotated -90.0 degrees.

h. Reset Statement

Syntax

```
<RESET STATEMENT>::=
    RESET <GRAPHICS/TEXT>
```

```
<GRAPHICS/TEXT>::=
    GRAPHICS
    |TEXT
    |GRAPHICS,TEXT
    |TEXT,GRAPHICS
```

Semantics

The Reset Statement serves to clear the graphics device of all text and/or graphics and initialize it for follow-on displays.

Examples:

```
    RESET TEXT
```

Text information is cleared from the screen.

i. Input Statement

Syntax

```
<INPUT STATEMENT>::=
    INPUT <DEVICE>

<DEVICE>::=
    PEDALS(<INTEGER ARRAY>)
    |SWITCHES(<INTEGER ARRAY>)
    |JOY(<REAL ARRAY>)
    |DIALS(<REAL ARRAY>)

<INTEGER ARRAY>::=
    INTEGER*ARRAY

<REAL ARRAY>::=
    REAL*ARRAY
```

Semantics

The Input Statement polls the status and/or values of the pedals, switches, joystick, or variable control dials.

Examples:

```
INPUT DIALS(DARRAY)
```

The setting of the variable control dials are stored in the array "DARRAY". (For the AGT-10, this array would be of length six, since there are six dials).

J. Output Statement

Syntax

```
<OUTPUT STATEMENT>:::=
    DISPLAY <PRIMITIVE/IMAGE LIST>
```

Semantics

The entities assigned to the listed text, primitive or image names are displayed on the graphics console.

Examples:

```
DISPLAY TITLE,CIRCUIT
```

The text string represented by "TITLE" is displayed on the graphics console according to its assigned line number, character position, size and intensity. The image represented by "CIRCUIT" is translated and rotated according to its transformation block and displayed in relation to the coordinate system set up by the coordinate specification statement. The primitives within the structure of "CIRCUIT" are displayed according to their attributes (i.e. MOVE/DRAW, DASHED/SOLID, and INTENSITY).

APPENDIX C DESCRIPTION OF SLIP

An implementation of GRAFTRAN on the XDS9300 utilizes Weizenbaum's SLIP [31] for its data structure. SLIP makes list processing facilities available to FORTRAN programs. An XDS9300 version of SLIP has been implemented by Miller [Ref. 32].

SLIP is a list processing system which is considered to be symmetric in the sense that its lists do not have a preferred orientation. That is, any operation which can be carried out on the top of a list can just as easily be carried out on the bottom. Symmetry is achieved by providing each list cell with a forward and a backward link as well as a datum.

Facilities are provided in SLIP to create and manipulate lists. Lists can be split or merged. Cells can be added or deleted at any point within a list. The contents of any list cell can be accessed or altered. Hierarchy or structure of list data can be achieved in SLIP by including list names within lists. Hence a complex data structure such as the one used in Sketchpad can be constructed. A major portion of SLIP/XDS9300 (hereafter simply referred to as SLIP) is implemented as machine independent FORTRAN routines. Machine dependent operations are performed by a few very primitive routines written in FORTRAN/Assembly-Language and are called on by the independent routines as required. This division of dependent and independent routines was created in Weizenbaum's original implementation of SLIP to facilitate transferability of the language. Essentially, the only routines requiring alternation when implementing SLIP on other computer configurations such as the XDS9300 are the primitives.

A complete description of SLIP can be found in N.P.S. Computer Laboratory Technical Note TN-1-3.

APPENDIX D PREPROCESSOR

The GRAFTRAN preprocessor writes a copy of all cards on an output file. Although any character may appear in column one, two characters are reserved for GRAFTRAN. One character ("G") is used to indicate that the statement is a GRAFTRAN statement which will cause the statement to be parsed for correct syntax. If the statement is correct, appropriate FORTRAN code will be produced and written on a code file. The second control character ("H") indicates that the GRAFTRAN program is complete and control is returned to the computer monitor system which will execute the FORTRAN program that was produced by GRAFTRAN. Since all cards are written on the output file as they are encountered, standard FORTRAN statements may be intermixed with GRAFTRAN statements.

Several routines of the preprocessor are machine dependent. These routines are concerned with packing a word or buffer or unpacking a word or buffer. An input card is temporarily stored in an eighty word buffer with one character (left justified) to a word. During syntax checking, this array is unpacked one character at a time until the statement is complete. Upon completion, correct FORTRAN code is packed into an eighty word buffer that is written on a code file.

The majority of the routines are concerned with analyzing each GRAFTRAN statement for correct syntax and emitting appropriate FORTRAN code to perform the indicated operation. If an error is detected, an error subroutine prints an appropriate message with a pointer to the erroneous code. The remaining portion of that statement is ignored. Three routines are initialization routines which print appropriate FORTRAN code to declare needed variables and dimension several arrays.

The FORTRAN code emitted is a "COMMON Statement" that is also used in the support routines (see Appendix E) which are needed when the program produced by GRAFTRAN is executed. If any modifications or routines are added to GRAFTRAN, care must be taken that the same "COMMON Statement" is included in any additional routines if the needed variables or arrays are contained in the COMMON Statement.

GRAFTRAN was implemented on an XDS9300 Computer with an ADAGE graphics terminal. A complete listing of GRAFTRAN and its support routines as implemented at the N.P.S. Computer Laboratory can be found in the Computer Laboratory Technical Note TN-13-3.

APPENDIX E RUN TIME ROUTINES

There are several routines which are required to execute the program produced by the GRAFTRAN preprocessor. Each of these routines can be put into one of three broad categories: (1) routines which are concerned with manipulating SLIP cells, (2) routines which are used to communicate with, or format data for the graphics console, and (3) routines which are needed to display text on the graphics console. The greater portion of the routines are written in FORTRAN, however, on a few occasions the requirement to manipulate a machine word arose and assembly language was used in these instances.

The routines used to manipulate SLIP cells, call "SLIP Subroutines" to create, delete, insert, and change data in SLIP cells to perform the operation indicated by the user. Several of these routines are called in the same order after an image or primitive is defined. This is done to set default values in the primitive structures and also to ensure that the correct number of cells for each structure is allocated in the correct order.

The routines required to display text on the graphics console include setting and retrieving several variables. These are as follows:

1. Character position on the scope.
2. Line number on the scope.
3. Length of the text string.
4. Size of the characters to be displayed.
5. Intensity of the characters.
6. Position of the text string in the text array.

The routines required to communicate with the graphics console must perform several functions. These are as follows:

1. Maintain text and graphics directories.
2. Reset text and graphics directories.
3. Format data to be transmitted to the graphics console.
4. Select graphics console to be used.
5. Set the coordinate scheme and scale factor.

These routines, which call several FORTRAN routines [Ref. 33] unique to the N.P.S. Computer Laboratory, are needed to execute the program created by the GRAFTRAN preprocessor which allows a user to display a structure on the graphics console, yet be completely isolated from the mechanism to do so. A complete listing and explanation of these routines can be found in N.P.S. Computer Laboratory Note TN-13-3.

```
      DIMENSION R(2,13),T(2,15),G(2,13),IS(16)
      DATA R/0.0,0.0,0.5,0.0,0.6,0.1,0.7,-0.1,0.8,0.1,0.9,-0.1,1.0,
     A0.1,1.1,-0.1,1.2,0.1,1.3,-0.1,1.4,0.1,1.5,0.0,2.0,0.0/
      DATA T/0.0,0.0,0.0,0.1,-0.5,0.5,0.0,0.1,0.0,0.3,0.0,-0.1,-0.375,
     A-0.4,-0.4,-0.375,-0.5,-0.5,-0.35,-0.425,-0.375,-0.4,0.0,-0.1,0.0,
     B-0.3,0.0,0.0,0.5,0.0/
      DATA G/0.0,0.0,0.0,-0.4,-0.3,-0.4,0.3,-0.4,0.0,-0.4,0.0,-0.6,
     A-0.2,-0.6,0.2,-0.6,0.0,-0.6,0.0,-0.8,-0.1,-0.8,0.1,-0.8,0.0,-0.8/
G     PRIMITIVE XISTOR,RES1,RES2,CON1,CON2,CON3,CON4,CON5,CON6,CON7,GND
      I=106711B
S      STA     07067
      IDEV=1
G     IMAGE TRL1,TRL2,TRL3,NOR
G     TEXT DESC,BY,PRES
G     DESC=(17,27,,)'DESCRIPTION OF A TRL THREE-INPUT NOR GATE'
G     TEXT GRAF
G     GRAF=(13,36,,)'GRAFTRAN SAMPLE PROGRAM'
G     BY=(21,25,,)'BY LT. DAVID R. ELKINS AND LT. EDWARD J. WOLF'
G     PRES=(31,30,,)'PRESS FUNCTION SWITCH 13 TO CONTINUE'
G     RESET TEXT
G     DISPLAY DESC,BY,PRES
G     DISPLAY GRAF
    5 CONTINUE
G     INPUT SWITCHS(IS)
      IF (IS(13).EQ.0) GO TO 5
G     TEXT COMP
G     COMP=(28,24,,)'COMPONENTS FOR A TRANSISTOR-RESISTOR-LOGIC UNIT'
G     PRES=(31,30,,)
G     XY(-5.0,-5.0,.666)
G     XISTOR=LIST(T,15)/(0.5,0.0)(-0.5,0.5)(-0.5,-0.5)/
G     RES1=LIST(R,13)/(0.0,0.0)(2.0,0.0)/
G     GND=LIST(G,12)/(0,0,0.0)/
G     CON1=LINE(0.0,-0.25)/(0.0,0.0)(0.0,-0.25)/
G     CON4= COPY CON1
G     CON2=COPY CON1
```

```
G      RES2= C0PY RES1
G      C0N3=LINE(4.0,0.0)/(0.0,0.0)(4.0,0.0)/
G      TRANSLATE(4.5,7.0) XIST0R
G      TRANSLATE(6.0,7.0) RES1
G      TRANSLATE(10.0,7.125) C0N1
G      RESET GRAPHICS,TEXT
G      DISPLAY C0MP,PRES
G      DISPLAY XIST0R,RES1,C0N1
   10  C0NTINUE
G      INPUT SWITCHS(IS)
       IF (IS(13).EQ.0) G0 T0 10
G      TEXT TRL
G      TRL=(30,20,,)'RTL UNIT AND C0MP0NENTS T0 BUILD A 3 INPUT N0R GATE'
G      PRES=(33,30,,)
G      TRL1=(XIST0R(3):C0N1(1))(XIST0R(1):RES1(1))(XIST0R(2):C0N2(2))
G     1/C0N2(1),C0N1(2),RES1(2)/
G      R0TATE (-90.0) C0N4,RES2
G      TRANSLATE(4.0,7.66) RES2
G      TRANSLATE(6.0,7.0) TRL1
G      TRANSLATE(10.0,7.5) GND
G      TRANSLATE(4.2,5.0) C0N3
G      TRANSLATE(9.7,5.0) C0N4
G      RESET TEXT,GRAPHICS
G      DISPLAY RES2,TRL1,GND,C0N4,C0N3
G      DISPLAY TRL,PRES
   20  C0NTINUE
G      INPUT SWITCHS(IS)
       IF (IS(13).EQ.0) G0 T0 20
G      TRL2=C0PY TRL1
G      TRL3=C0PY TRL1
G      C0N5=C0PY C0N3
G      C0N6=C0PY C0N3
G      C0N7=C0PY C0N3
G      TEXT N0RG,C0N,TECH,ANY,0UT,0NLY,INPU
G      TEXT 0UTPU,LABL
```

```
G     NORG=(30,24,,)'THE THREE-INPUT NOR GATE IS CONSTRUCTED USING TRL'
G     CON=(31,24,,)'(CONTRACTION OF TRANSISTOR-RESISTOR-LOGIC)'
G     TECH=(32,24,,)'TECHNOLOGY, TRL SIMPLY MEANS THAT ONLY TRANSISTORS'
G     ANY=(33,24,,)'AND RESISTORS ARE UTILIZED IN THE LOGIC UNIT. THE'
G     OUT=(34,24,,)'OUTPUT OF THE THREE-INPUT NOR GATE IS A LOGICAL ONE'
G     ONLY=(35,24,,)'ONLY WHEN A LOGICAL ZERO IS ON EACH OF THE THREE'
G     INPU=(36,24,,)'INPUTS, A, B, AND C.'
G     LABL=(22,37,,)'A                        B                        C'
G     OUTPU=(20,11,,)'OUTPUT'
G     RESET TEXT
G     DISPLAY NORG,CON,TECH,ANY,OUT,ONLY,INPU
G     DISPLAY LABL,OUTPU
G     NOR=(TRL1(1):CON3(1))(TRL1(2):CON5(1))(TRL1(1):CON6(2))
G    1      (TRL1(2):CON7(2))(TRL1(1):RES2(2))(TRL1(2):GND(1))
G    2      (CON6(1):TRL2(1))(CON3(2):TRL3(1))(TRL2(1):CON4(1))
G    3      /TRL2(3),TRL1(3),TRL3(3),CON4(2),RES2(1)/
G     TRANSLATE(6.5,7.0) NOR
G     RESET GRAPHICS
G     DISPLAY NOR
      STOP
      END


H
```

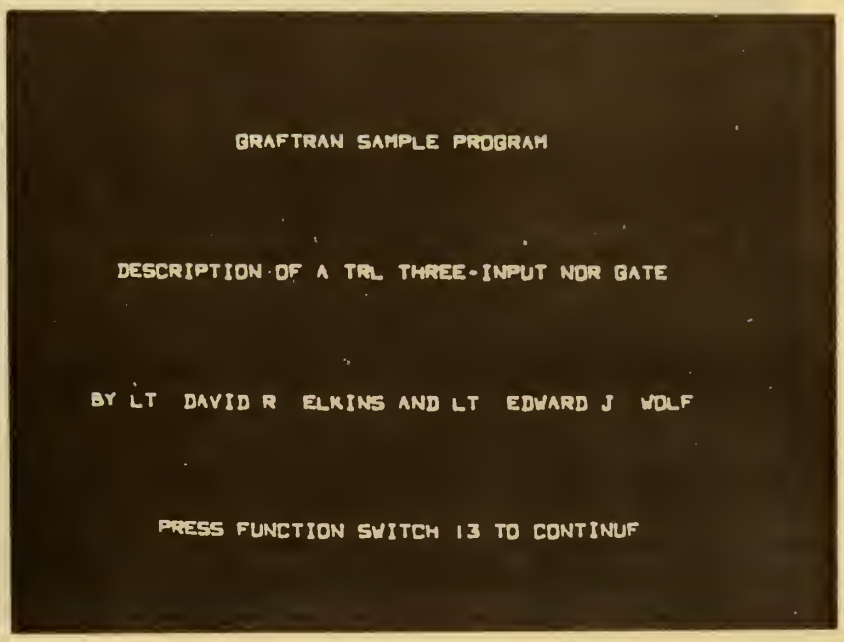


FIGURE 10--DISPLAY FRAME 1

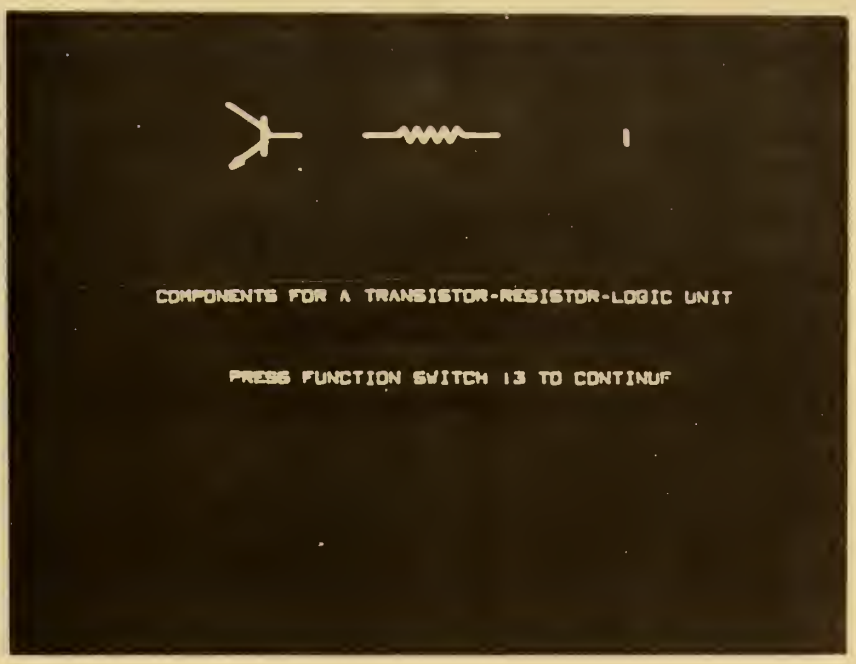


FIGURE 11--DISPLAY FRAME 2

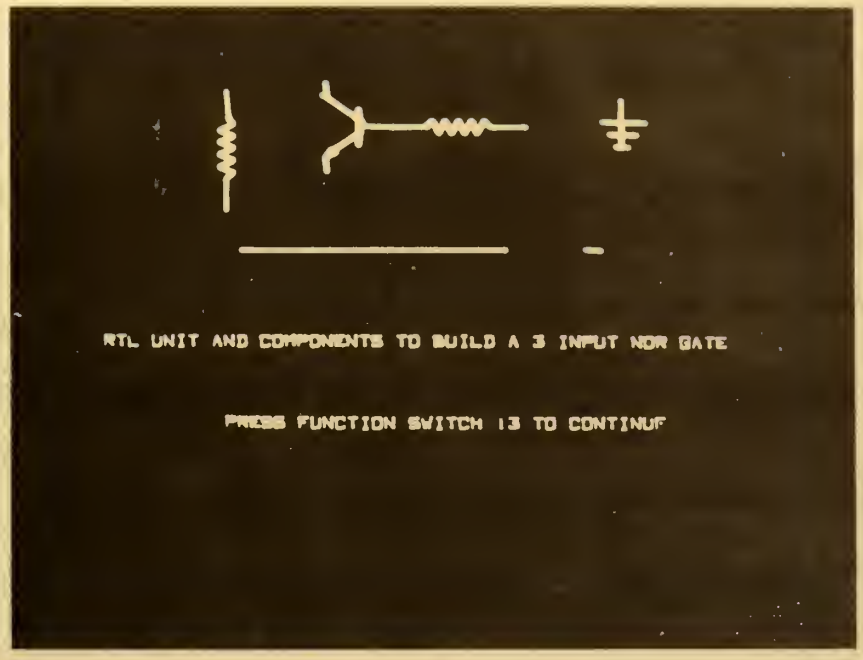


FIGURE 12--DISPLAY FRAME 3

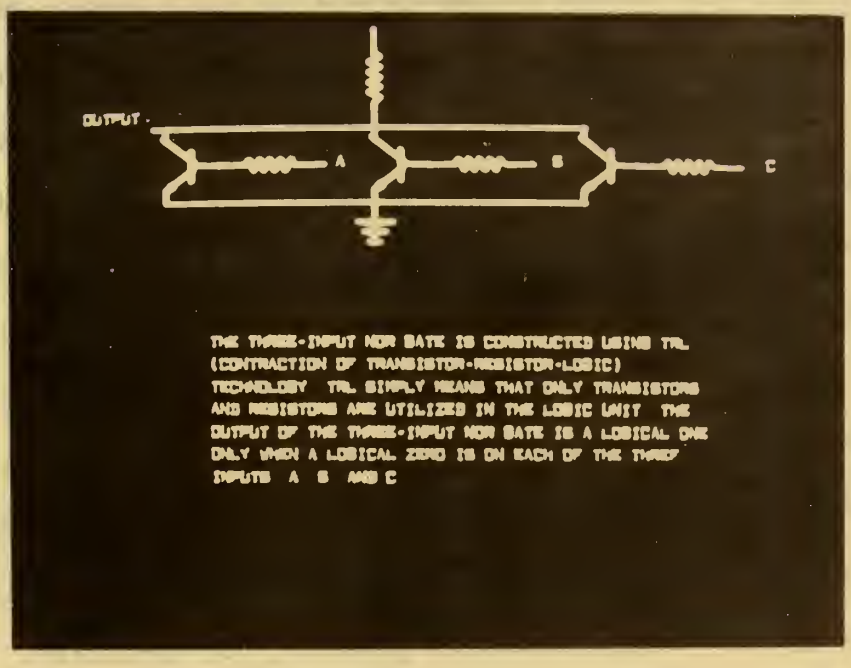


FIGURE 13--DISPLAY FRAME 4

APPENDIX G FUTURE DEVELOPMENT

This Appendix contains suggestions for future development of GRAFTRAN. Not all of the suggestions are useful to all program applications. The item that the authors wish to stress is the interactive capability at the graphics console which is not only useful but needed in a truly interactive environment.

One interaction feature is an automatic menu select. This could be implemented by developing a command, "DISPLAY MENU", and allowing the user to select an item on the menu via light pen or teletypewriter. This would cause branching to occur to the appropriate position in the program as indicated by the selected item. This would require allowing input from the light pen or the teletypewriter which as yet has not been developed. Input from the teletypewriter should be of the same form as the FORTRAN "READ STATEMENT". This would allow the computer to decode any information the user desired to input into the system.

Another feature that could be added is a connect operator which could be used at the graphics console. This would allow the user to select two disjoint points displayed on the console, and connect them with a line or in the manner specified by the user.

There are several non-interactive features which could be developed that would allow the user greater flexibility in the construction and manipulation of primitives, images, and text. These are as follows:

A. Primitives for producing ellipses, spline curves, etc., should be added.

B. A modification of the list primitive which would allow a master move-draw list such that the MOVE-DRAW option can be

selectively interspersed within the list as specified by the user.

C. The ability to dimension primitives, images, and text which would allow the user to subscript into the memory space occupied by a primitive, image, or text.

D. A description list which would allow a user to tag an element with a value (or values). For example a resistor could be tagged with a value of 15 ohms and the title of a text declaration "RESISTOR". When the user displayed the resistor, the position of the text "RESISTOR" would automatically be calculated and displayed on the console next to the resistor. The value of 15 ohms would remain within the resistor to be used in the analysis of the image.

E. An erase operation which would replace the selected name and storage associated with that name back to the dynamic free storage list.

F. A reflection operator which would produce a "MIRROR IMAGE" of a declared primitive or image. This would be especially useful in symmetric graphics applications such as circuit design.

G. An axis scale definition which would allow a different scale factor to each axis.

H. A plot feature which would be useful to obtain a "HARD COPY" of whatever is displayed on the console. Whether the copy would be printed on the line printer, cal-comp plotter, or whatever equipment is available, would be selectively declared by the user.

I. String operators which would allow; (1) the ability to concatenate declared text strings, (2) the ability to compare text strings, and (3) the ability to use substring operations.

J. The ability to declare a logarithmic, polar, or three-dimensional axis system.

Two modifications to GRAFTRAN would allow greater flexibility in the FORTRAN program that is produced. The first modification would allow labels on a GRAFTRAN statement. This would allow "LOOPING" within any portion of the produced program or branching to any position in the program. In GRAFTRAN's present form, only real or integer values (or variables) are allowed as parameters in a GRAFTRAN statement. The second modification to implement would be to allow arithmetic expressions as a parameter.

Some modifications could be made to GRAFTRAN which would be unique to the N.P.S. Computer Laboratory. Also some features of the N.P.S. Computer Laboratory system could be modified to enhance GRAFTRAN's capabilities. One modification of GRAFTRAN would be "OVERLAYING" the routines which are required to execute the programs produced by GRAFTRAN. Since only 32k of storage is available, large and complex programs could quickly use all the available storage. This could be partially solved by putting the support routines on the procedure library: thus, only the required routines would be loaded into main memory. A paging mechanism for text space could also be developed so that the text array would not need to be in the main memory unless demanded, in which case the desired text would be paged into main memory.

A modification to the N.P.S. Computer Laboratory system would allow random access files on the XDS9300. In its present implementation,

efficiency is greatly reduced when page zero is filled. When this modification is incorporated into the computer system, the procedure to implement the change to SLIP can be found in [Ref. 32].

A modification to the "GATED" program of the graphics terminal would alter the method of maintaining the graphics and text directories. In its present implementation, items are placed in the text and graphics directories in a sequential manner. Also the text directory is maintained using the line number and character position information of the text as an index into the text directory. Duplication of line number and character position information is prohibited. It would be helpful to simply declare items to be placed in specific positions in either directory and consequently display items by "NAME" or position in the directory. This would also allow a statement of the form "DISPLAY TEXT or GRAPHICS(1-5)" which would display the items in the specified directory from one to five.

The modifications above would make GRAFTRAN a very powerful language, yet extremely easy to use and learn. This would be beneficial to a user that is not familiar with computers but wants to use a computer graphics system to aid him in solving a problem.

BIBLIOGRAPHY

1. Licklider, J.C.R., "Man-Compter Symbiosis," IRE, Transactions HFE, pp. 4-11, March 1960.

2. Computer Science Department, Stanford University Report STAN-CS-72-306, A Bibliography on Computer Graphics, by B. W. Pollack, August 1972.

3. Johnson, C.I., "Principles of Interactive Systems," IBM Systems Journal, Nos. 3 and 4, pp. 147-173, 1968.

4. Sutherland, I.E., "Sketchpad a Man-Machine Graphical Communication System," AFIPS Conference Proceedings 1963, Spring Joint Computer Conference, v. 23, Spartan Bocks Incorporated, 1963.

5. Johnson, T.E., "Sketchpad III a computer Program for Drawing in Three Dimensions," Proceedings of the 1963 Spring Joint Computer Conference, pp. 347-353.

6. Rully, A.D., "A Subroutine Package for FORTRAN," Interactive Graphics in Data Processing, IBM Systems Journal, v. 7, pp. 248-256, 1968.

7. The RAND Corporation Report RM-5531-ARPA, The Integrated Graphics System for the IBM 2250, by G.D. Brown and C.H. Bush, 1968.

8. Mezei, L., "SPARTA, A Procedure Oriented Programming Language for the Manipulation of Arbitrary Line Drawings," Information Processing 68, North-Holland Publishing Company, pp. 597-604, 1969.

9. Hurwitz, A., and Others, "GRAF: Graphic Additions to FORTRAN," AFIPS Conference Proceedings 1967, Spring Joint Computer Conference, v. 30, Thompson Book Company, 1967.

10. The RAND Corporation Report RM-6028-ARPA, Extensions to the PL/1 Language for Interactive Graphics, by R.H. Anderson and D.J. Farber, January 1970.

11. Kulsrud, H.E., "A General Purpose Graphic Language," Communications of the ACM, v. 11, pp. 247-254, April 1968.

12. Stanford Linear Accelerator Center, Stanford University Report SLAC-134, GEMS-A Graphical Experimental Meta System, by J.E. George, p. 184, August 1971.

13. Stanford Linear Accelerator Center, Stanford University Report SLAC-134, SIMPLE-A Simple Precedence Translator Writing System, by J.E. George, 1971.

14. George, J.E. and Miller, W.F., "String Description of Data for Display," 9th National Symposium on Information Display, pp. 143-147, 1968.

15. The RAND Corporation Report RM-5825-PR, POGO: Programmer-Oriented Graphics Operation, by B.W. Boehm, V.R. Lamb, R.L. Mobley, and J.E. Rieber, p. 44, June 1967.

16. Bechtel, W.D., On-Line Graphical Display System, Master's Thesis, Naval Postgraduate School, p. 130, Monterey, June 1968.

17. Mayer, S.H., SCOPE: An Introductory Graphics Language, Master's Thesis, Naval Postgraduate School, p. 98, Monterey, 1964.

18. Beans, J.D., GPGL: A Model Interactive, General Purpose Graphic Language, Master's Thesis, Naval Postgraduate School, p. 117, Monterey, December 1971.

19. Chasen, S.H., "Experience in the Application of Interactive Graphics," in Emerging Concepts in Computer Graphics, Secrest, D., and Nievergelt, J., ed., W.A. Benjamin Incorporated, pp. 255-309, 1968.

20. Baskin, M.B. and Morse, S.P., "A Multilevel Modeling Structure for Interactive Graphics," IBM Systems Journal, Nos. 3 and 4, pp. 218-228, 1968.

21. Sibley, E.H., "The Use of a Graphic Language to Generate Graphic Procedures," in Pertinent Concepts in Computer Graphics, Faiman, M., and Nievergelt, J., ed., University of Illinois Press, pp. 390-413, 1968.

22. Baskin, H.B., "A Comprehensive Application Methodology for Symbolic Computer Graphics," in Pertinent Concepts in Computer Graphics, Fairman, M., and Nievergelt, J., ed., University of Illinois Press, pp. 414-428, 1968.

23. Wolf Research and Development Corporation Report NASA CR-1628, Interactive Specification of Data Displays, by R. Anderson and others, June 1970.

24. SDS FORTRAN IV, Scientific Data Systems Reference Manual, Scientific Data Systems, Santa Monica, Cal. 1967.

25. SDS 9300 Computer Manual, Scientific Data Systems Reference Manual, Scientific Data Systems, Santa Monica, Cal. 1967.

26. Knowlton, K.C., "A Programmer's Description of L6," Communications of the ACM, v. 9, pp. 616-625, August 1966.

27. Rovner, P.D. and Feldman, J.A., "The Leap Language and Data Structure," Proceedings of IFIP Congress 1968, pp. 579-585.

28. The RAND Corporation Report RM-6145-ARPA, A Survey of Data Structures for Interactive Graphics, by J.A. Hamilton, April 1970.

29. Lang, C.A. and Gray, J.C., "ASP-A Ring Implemented Associative Structure Package," Communications of the ACM, v. 11, pp. 550-555, August 1968.

30. Dodd, G.G., "APL-A Language for Associative Data Handling in PL/1," Proceedings - Fall Joint Computer Conference, 1966, pp. 677-684.

31. Weizenbaum, J., "Symmetric List Processor," Communications of the ACM, v. 9, pp. 524-536, September 1963.

32. Miller, L.H., A Graphics-Applications Data Structure for Medium Scale Computers, Master's Thesis, Naval Postgraduate School, p. 77, Monterey, 1970.

33. Delaura, R.D., Electrical Engineering Computer Laboratory Manual for the FORTRAN User, U.S. Naval Postgraduate School, Monterey, California, 1972.

34. Gries, D., Compiler Construction for Digital Computers, pp. 18-23, Wiley, 1971.

35. Naur, P., "Revised Report on the Algorithmic Language ALGOL 60," Communications of the ACM, v. 6, pp. 1-17, January 1963.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Professor G. L. Barksdale, Code 72 Chairman, Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	Professor G. E. Heidorn, Code 55Hd Naval Postgraduate School Monterey, California 93940	1
5.	N.P.S. Computer Laboratory Naval Postgraduate School Monterey, California 93940	2
6.	LT D. R. Elkins Hunters Point, Naval Shipyard San Francisco, California 94135	1
7.	LT E. J. Wolf 215 2nd Avenue Enderlin, North Dakota 58027	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY *(Corporate author)*

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

REPORT TITLE

GRAFTRAN: Graphic Extensions to FORTRAN

DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Master's Thesis; December 1972

AUTHOR(S) *(First name, middle initial, last name)*

David R. Elkins
Edward J. Wolf

REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1972	89	35

CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
PROJECT NO.	
	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

ABSTRACT

This paper describes a computer graphics language which can be used to easily display complex structures on a medium scale computer system, such as the XDS9300/AGT-10 complex at the Naval Postgraduate School. GRAFTRAN (Graphic Extensions to FORTRAN) provides the full capabilities of FORTRAN and includes features that represent a cross-section of operations which are used in present day computer graphics systems. The user is completely isolated from the mechanism to construct and display an image and the manipulation of the data involved. Also included is a survey of languages and data structures currently used in graphical applications. An example GRAFTRAN program is included.

DD FORM 1 NOV 68 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Graphics						
Graphics Language						
Graphics Data Structure						
Preprocessor						
SLIP						